VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Automatické parkování automobilu

Automatic Car Parking System

2017 Luboš Matejčík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Luboš Matejčík**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Automatické parkování automobilu Automatic Car Parking System
Jazyk vypracování:	čeština

Zásady pro vypracování:

Navrhněte parkovací systém modelu automobilu s Ackermanový podvozkem, který je vybaven laserovým skenerem pro měření vzdáleností od překážek. Parkovací systém musí být schopen rozpoznat, zda parkovací místo je vhodné pro podélné či kolmé parkování a následně provést potřebné úkony pro realizaci samotného zaparkování.

1. Seznamte se s řízením Ackermanova podvozku.
2. Seznamte se s laserovým skenerem Hokuyo URG-04lx, navrhněte a otestujte detekci objektů v okolí.
3. Seznamte se modelem auta, jeho rozměry a způsobem řízení.
4. Navrhněte matematický model pro kolmé i podélné parkování.
5. Aplikujte navržený model parkování na modelu auta v uměle vytvořeném prostředí.
6. Otestujte a vyhodnoťte spolehlivost parkování pro různě velká parkovací místa.

Seznam doporučené odborné literatury:

[1] JS Zhao, X Liu, ZJ Feng, JS Dai, Design of an Ackermann Type Steering Mechanism, https://www.researchgate.net/publication/265755401_Design_of_an_Ackermann_Type_Steering_Mechanism
[2] JM Snider, Automatic steering methods for autonomous automobile path tracking, https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf
[3] A Gupta, R Divekar, Autonomous Parallel Parking Methodology for Ackerman Configured Vehicles, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.394.2446&rep=rep1&type=pdf
[4] A Schaub, JCR de la Cruz, D Burschka, Autonomous Parking using a Highly Maneuverable Robotic Vehicle, http://www6.in.tum.de/Main/Publications/Schaub14.pdf
[5] MFB Wahab, AL Moe, AB Abu, ZB Yaacob, A Legowo, DEVELOPMENT OF AUTOMATED PARALLEL PARKING SYSTEM IN SMALL MOBILE VEHICLE, http://www.arpnjournals.com/jeas/research_papers/rp_2015/jeas_0915_2526.pdf
[6] N Shaha, N Nakranib, Autonomous Parking System Techniques – A Review,

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval svému vedoucímu práce Ing. Petru Olivkovi, Ph.D. a jeho kolegům za jejich cenné rady a velmi užitečnou pomoc.

Abstrakt

Tato práce pojednává o řízení modelu automobilu s Ackermannovým podvozkem k řešení úlohy automatického parkování na kolmé a podélné parkovací místo. Pro řešení úlohy orientace robota v prostoru a mapování je použit algoritmus ICP-SLAM. V práci je použit laserový měřič vzdáleností Hokuyo URG-04LX. Jako hlavní řídící počítač je použit Raspberry Pi 2. U automatického parkování pro kolmá parkovací místa bylo docíleno 93% úspěšnosti. U automatického parkování pro podélná parkovací místa byla úspěšnost vyhodnocena na 76 %.

Klíčová slova: LIDAR, SLAM, ICP, mřížka obsazenosti, Ackermannova geometrie podvozku, metoda nejmenších čtverců, Raspberry Pi, Hokuyo URG-04LX, FRDM-K64F, FRDM-TFC

Abstract

This paper deals with driving a car model with Ackermann steering geometry for automatic parking on the perpendicular and parallel parking place. The algorithm ICP-SLAM is used to solve the robot's orientation task in space and mapping. The Hokuyo URG-04LX laser range finder is used in the work. The Raspberry Pi 2 is the main control computer. With automatic parking for perpendicular parking places, 93% of success was achieved. For automatic parking for parallel parking places the success rate was evaluated at 76%.

Key Words: LIDAR, SLAM, ICP, occupancy grid, Ackermann steering geometry, Least squares, Raspberry Pi, Hokuyo URG-04LX, FRDM-K64F, FRDM-TFC

Obsah

Seznam použitých zkratek a symbolů

FRDM	–	Freedom Development Platform
LIDAR	–	Light Detection And Ranging
USB	–	Universal Serial Bus
STX	–	Start of text
ETX	–	End of text
PWM	–	Pulse Width Modulation – pulzní šířková modulace
SLAM	–	Simultaneous Localization and Mapping – simultánní lokalizace a mapování
ICP	–	Iterative Closest Point
LRF	–	Laser Range Finder – laserový měřič vzdálenosti

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílém mé diplomové práce je vytvořit autonomní řízení modelu automobilu s Ackermannovým podvozkem, které řídí automobil tak, aby automaticky zaparkoval na kolmé či podélné parkovací místo.

Autonomní řízení automobilů se v dnešní době velice rozrůstá a společnosti zabývající se touto problematikou již nasazují vozidla, která umožňují autonomní řízení počítačem. Tyto řídící systémy musí být dokonalé a bezchybné. Jejich vyhodnocování nesmí selhat, vzhledem k tomu, že jsou v sázce životy cestujících. Stále se jedná o problematiku, vůči které jsou lidé skeptičtí, ale věřím, že bude brzy přijata širokou veřejností.

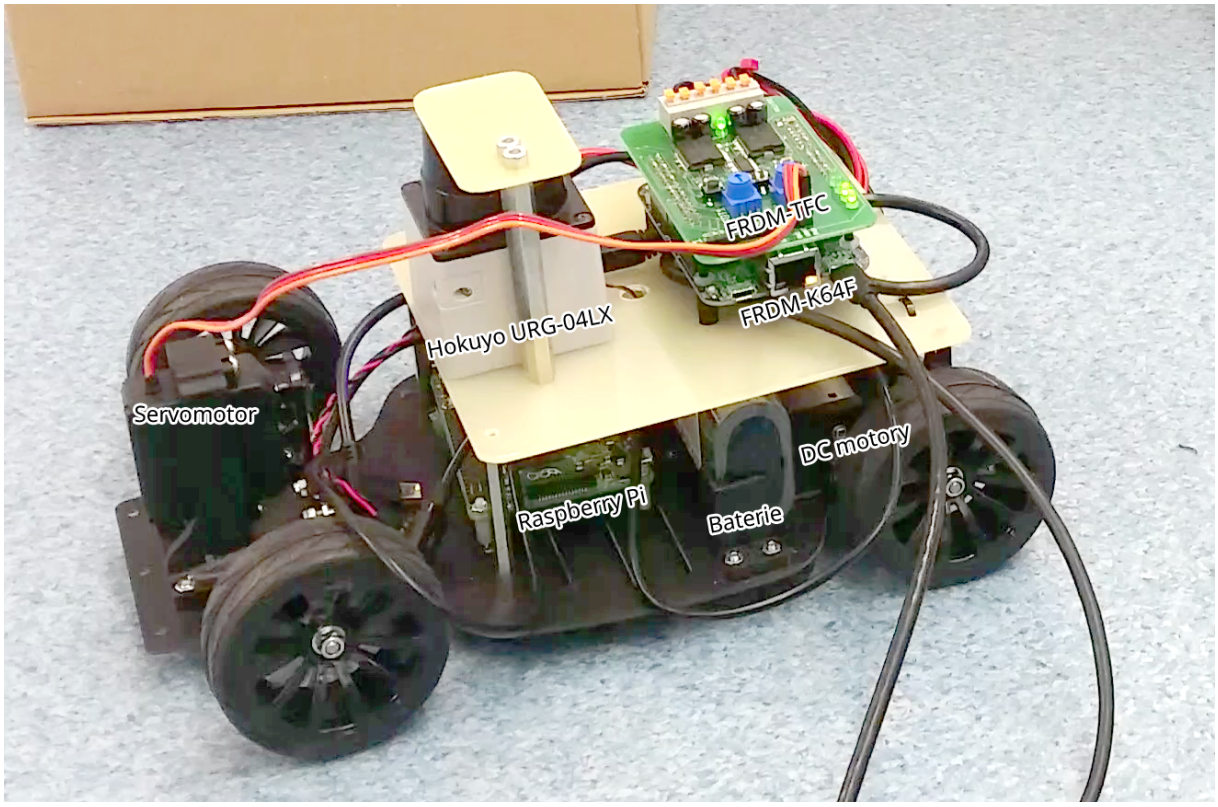
Pokud by na cestách jezdila pouze autonomní vozidla, neexistovaly by dopravní zácpy a nehody. Řidiči by mohli čas strávený na cestách využít jinak, než soustředěním se na řízení. Díky centrální komunikaci všech autonomních vozidel, by bylo vyhodnocování nebezpečných stavů na cestách okamžité a každé autonomně řízené vozidlo by reagovalo již s předstihem. Lidský faktor, který způsobí většinu dopravních nehod bude minulostí. Kolony, které jsou tvořeny díky reakční době člověka, z cest úplně zmizí. Tato představa je již blízká budoucnost.

2 Model automobilu

V této práci je použit model automobilu o rozměrech 19×28 cm s geometrií řídícího mechanismu podle Ackermannovy podmínky s náhonem na zadní kola. Rozvor náprav je 19 cm a rozchod předních kol je 13,5 cm. Na obrázku 1 je použitý model automobilu s potřebnými moduly k ovládání motorů a se skenerem pro měření vzdáleností od překážek Hokuyo URG-04LX, který využívá technologii LIDAR.

Natáčení předních kol je ovládáno servomotorem. Zadní kola jsou poháněna dvěma elektrickými motory.

K modelu automobilu je upevněna sklotextitová deska. Na vrchní straně desky je připevněn skener pro měření vzdáleností od překážek, společně s řídícím modulem FRDM-K64F a silovou částí s označením FRDM-TFC. Na spodní straně sklotextitové desky je upevněna svorkovnice pro připojení baterie a stejnosměrný měnič napětí, který obstarává napětí 5 V pro skener a hlavní počítač Raspberry Pi, který je umístěn na vrchní straně podvozku modelu.

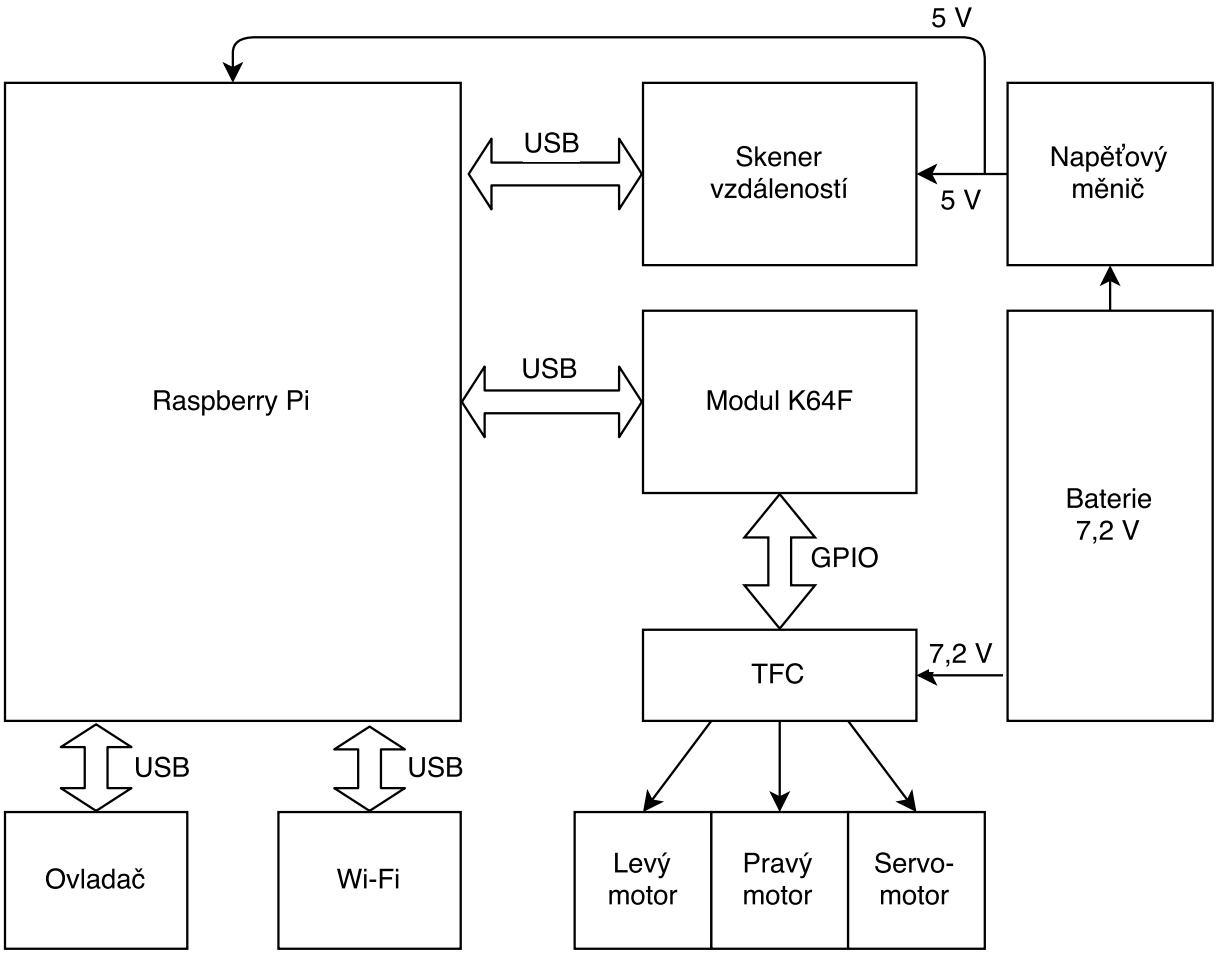


Obrázek 1: Model automobilu s potřebnými moduly a skenerem

3 Řídící moduly a hlavní počítač

Raspberry Pi 2 Model B byl zvolen jako hlavní počítač vzhledem k jeho relativně velké paměti o velikosti 1 GB a dostatečně rychlému (900 MHz) čtyřjádrovému procesoru ARM Cortex-A7. V Raspberry Pi je použita paměťová karta microSD s kapacitou 16 GB.

Diagram zapojení je uveden na obrázku 2.



Obrázek 2: Diagram zapojení prvků modelu automobilu

3.1 Hlavní počítač Raspberry Pi

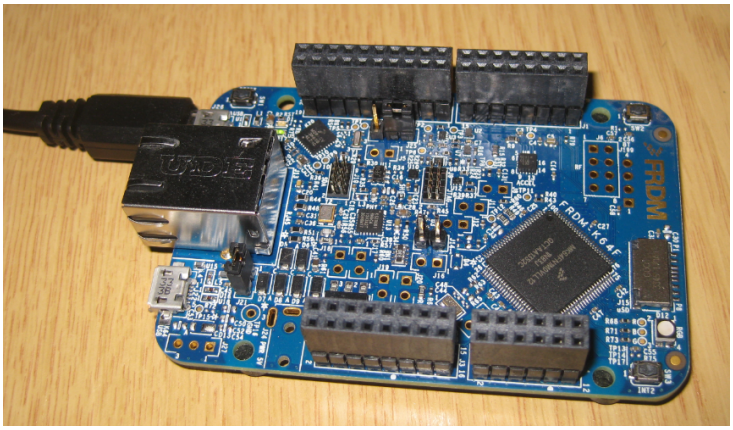
Na hlavním počítači je nainstalován operační systém Minibian. Jedná se o upravený operační systém Debian pro Raspberry Pi bez grafického uživatelského rozhraní a s minimálním množstvím instalovaných balíků, aby byla velikost operačního systému co nejmenší.

K Raspberry Pi je připojen přes USB rozhraní herní ovladač s páčkami pro ovládání automobilu. Dále je k Raspberry Pi připojen Wi-Fi adaptér, který pracuje v režimu přístupového bodu a vytváří tak možnost bezdrátového připojení.

Poslední dvě klíčová zařízení připojena k Raspberry Pi jsou modul FRDM-K64F a skener vzdáleností od překážek.

3.2 Modul FRDM-K64F

Jedná se o vývojovou desku od společností NXP Semiconductors s mikrokontrolérem postaveným na ARM Cortex-M4 jádru MK64FN1M0VLL12, viz obrázek 3. Deska je vybavena dvěmi USB rozhraními s micro USB konektory, ethernetovým rozhraním, paticí pro microSD kartu, GPIO piny. Celkové blokové schéma je na obrázku 4 [1].



Obrázek 3: Modul FRDM-K64F

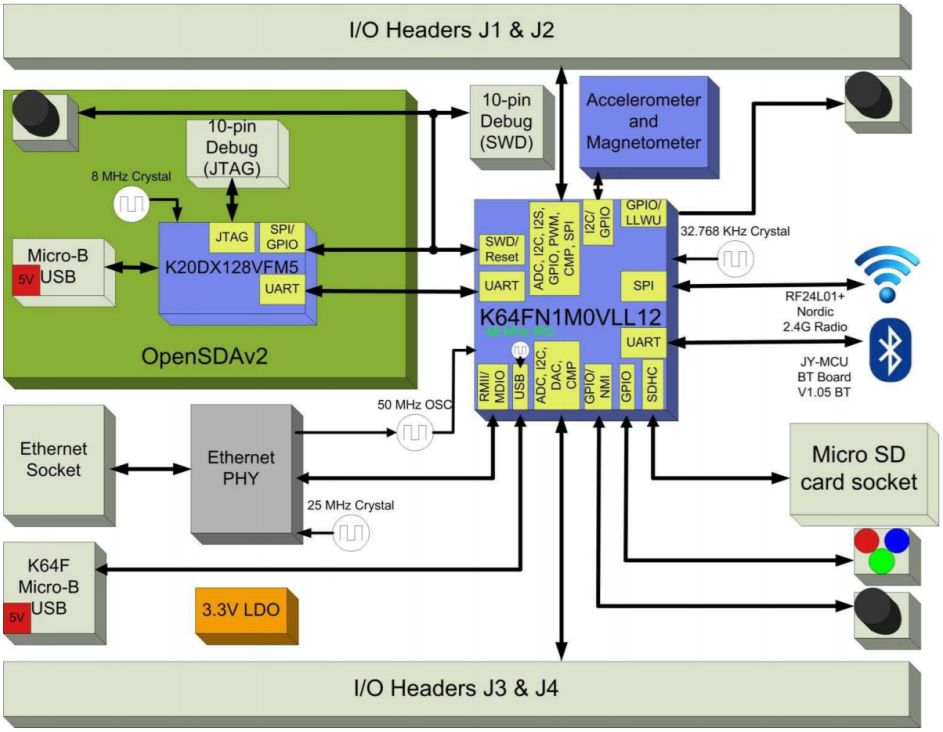
Modul FRDM-K64F má v sobě nahrán firmware, který obstarává ovládání automobilu. Raspberry Pi komunikuje s modulem FRDM-K64F přes rozhraní USB virtuální sériovou linkou.

3.2.1 Komunikační protokol

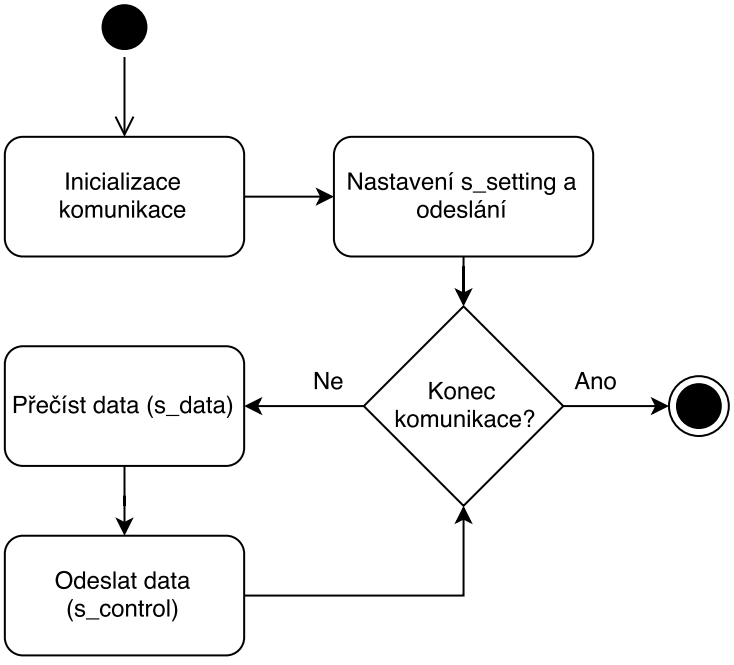
Přes sériovou linku se odesílají, resp. přijímají tři typy datových struktur. Tyto struktury jsou vždy obaleny doplňujícími informacemi pro přenos. V tabulce 2 je zobrazen datový rámec. `STX` a `ETX` jsou speciální bajty, které určují začátek a konec datového rámce. Po `STX` bytu následují dva bajty, které udávají délku přenášené sekvence. `CMD` bajt určuje o jaký typ přenášené struktury se jedná.

V komunikaci figurují tři typy datových struktur. Pro čtení dat slouží struktura `s_data`, nastavení modulu zajišťuje struktura `s_setting` a pro ovládání automobilu struktura `s_control`. Definice těchto struktur je uvedena ve výpise 1.

Struktury `s_setting` a `s_control` jsou odesílány z hlavního počítače a přijímány modulem. Struktura `s_data` je odesílána z modulu a přijímána hlavním počítačem.



Obrázek 4: Blokový diagram vývojové desky FRDM-K64F



Obrázek 5: Diagram aktivit komunikace s modulem FRDM-K64F

Tabulka 1: Detaily struktur

s_data						
člen	timestamp	adc	dip_sw	push_sw	image	_padding
typ	uint32	uint16[5]	uint8	uint8	uint16[128]	uint32
délka	4	10	1	1	256	4

s_setting				
člen	servo_center	servo_max_lr	pwm_max	_padding
typ	uint16[2]	uint16[2]	uint16	uint16
délka	4	4	2	2

s_control							
člen	leds	pwm_onoff	servo_onoff	_padding1	pwm_a	pwm_b	servo_pos
typ	uint8	uint8	uint8	uint8	int16	int16	int16[2]
délka	1	1	1	1	2	2	4

Tabulka 2: Popis komunikačního protokolu

	STX	**Délka**	**CMD**	**Data struktury**	**ETX**
Počet bajtů	1 bajt	2 bajt	1 bajt	x bajtů	1 bajt
Hodnota	0x02	LO bajt HI bajt	0x01 - 0x03	...	0x03

Kroky komunikace jsou znázorněny v diagramu aktivit na obrázku 5 a jsou následující:

1. Hlavní počítač incializuje komunikaci se zařízením.
2. Odešle naplněnou strukturu `s_setting`.
3. Přijme datový rámec a naplní strukturu `s_data` přijatými daty.
4. Odešle do modulu naplněnou strukturu `s_control`.
5. Proces se opakuje od bodu 3, pokud má komunikace nadále probíhat.

Detailní popis struktur je v tabulce 1.

3.2.2 Popis datové struktury s_data

- `timestamp`: Pořadové číslo datového rámce.
- `adc[0]`: Hodnota natočení potenciometru 1 na desce FRDM-TFC.
- `adc[1]`: Hodnota natočení potenciometru 2 na desce FRDM-TFC.
- `adc[2]`: Hodnota proudu protékajícího motorem pravého kola.
- `adc[3]`: Hodnota proudu protékajícího motorem levého kola.
- `adc[4]`: Hodnota napětí baterie.
- `dip_sw`: Stav přepínačů na desce FRDM-TFC. Bity 0 až 3.

- `push_sw`: Stav dvou tlačítek na desce FRDM-TFC. Bity 0 a 1.
- `image`: Data z řádkové kamery. 128 pixelů v hodnotách `uint16`.
- `_padding`: Rezerva pro případné rozšíření.

3.2.3 Popis datové struktury s_setting

- `servo_center`: Nastavení středové hodnoty natočení kol, od které se pak počítají hodnoty ±1000.
- `servo_max_lr`: Nastavení maxima natočení kol.
- `pwm_max`: Nastavení maxima PWM pro ovládání motorů.
- `_padding`: Rezerva pro případné rozšíření.

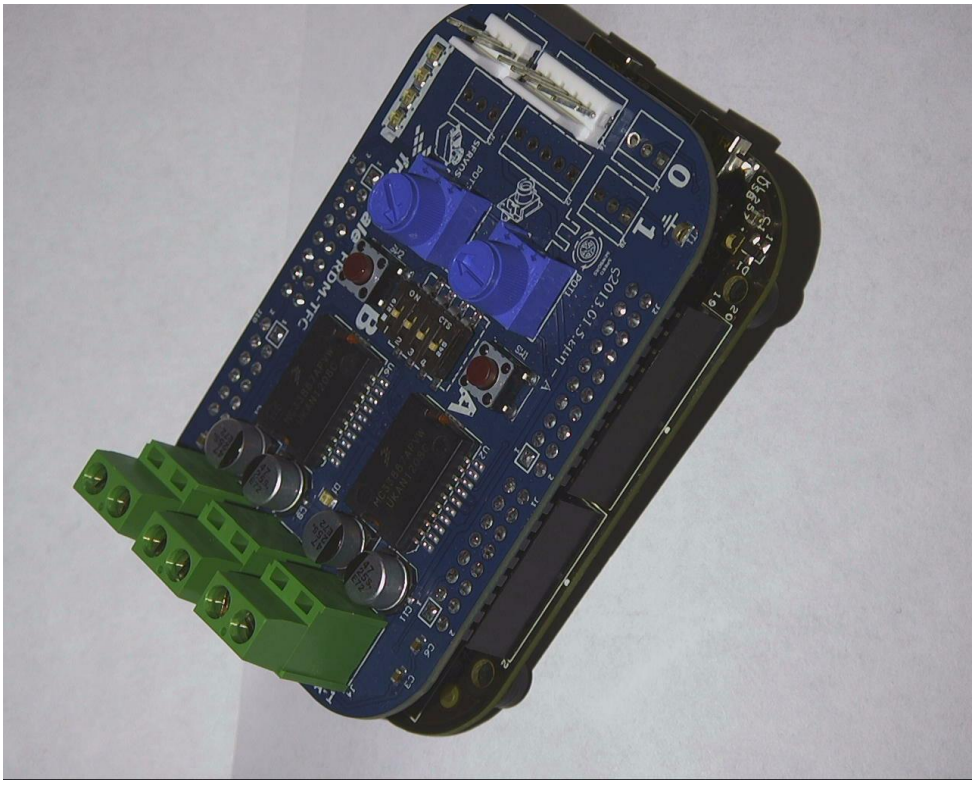
3.2.4 Popis datové struktury s_control

- `leds`: Určuje stav LED na desce FRDM-TFC. Bity 0-3.
- `pwm_onoff`: Vypnutí/zapnutí H-můstků. Bit 0 odpovídá motoru pravého kola. Bit 1 odpovídá motoru levého kola.
- `servo_onoff`: Vypnutí/zapnutí servomotoru.
- `_padding1`: Rezerva pro případné rozšíření.
- `pwm_a`: Nastavení hodnoty PWM pro pravé kolo.
- `pwm_b`: Nastavení hodnoty PWM pro levé kolo.
- `servo_pos`: Nastavení natočení servomotoru. Hodnoty ±1000.

3.3 Modul FRDM-TFC

Na obrázku 6 je modul FRDM-TFC připojen do GPIO svorkovnice modulu FRDM-K64F. Slouží pro ovládání dvou stejnosměrných motorů (až do 5 A na motor) a dvou servomotorů. Součástí modulu jsou H-můstky, ochranné obvody, svorkovnice, dva ponteciometry, spínací a přepínací tlačítka, 4 signalizační LED [3].

H-můstky MC33887 umožňují ovládání dvou stejnosměrných motorů, které slouží jako pohon automobilu, a také poskytují informace o proudech tekoucích do motorů, které jsou snímány AD převodníky.



Obrázek 6: Modul FRDM-TFC

4 Ackermannův podvozek

Podvozek automobilů bývá zpravidla sestaven ze tří nebo více kol, přičemž jedním či více koly lze natáčet a měnit tak směr jízdy. Kola, která se natáčí, mohou být použita také pro pohon.

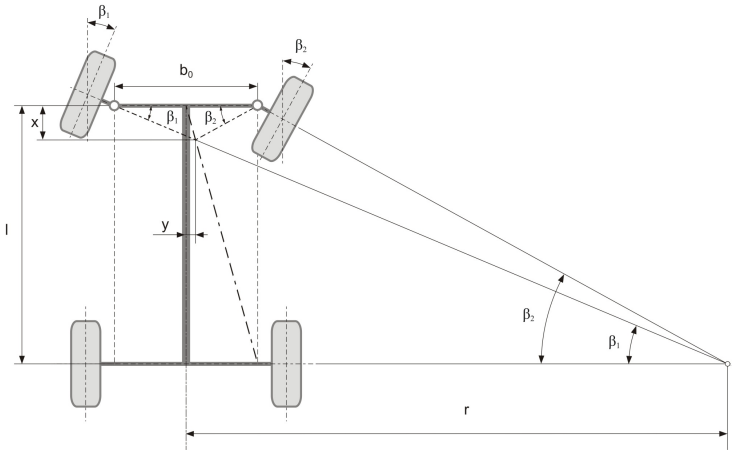
Při jízdě zatáčkou zajišťuje Ackermannův podvozek natočení pravého a levého kola v trochu jiném úhlu. Vnitřní a vnější kola opisují při jízdě zatáčkou kružnice o jiných poloměrech. Díky tomuto způsobu zatáčení je zamezeno nežádoucímu smýkání kol po povrchu. Ackermannova geometrie podvozku určuje střed otáčení na prodloužené ose zadní nápravy viz obrázek 7 [4, 10].

U automobilů, které využívají Ackermannův povozek, lze vyjádřit maximální úhel natočení kol pomocí nejmenšího poloměru otáčení.

Ackermannovu podmínku lze vyjádřit matematicky jako:

$$\cot \beta_1 - \cot \beta_2 = \frac{b_0}{l} \tag{1}$$

[11]



Obrázek 7: Geometrie Ackermannova podvozku

4.1 Poloměry otáčení u použitého modelu automobilu

Použitý model automobilu zatáčí podle bezrozměrné hodnoty `servo_pos` (±1000). Tato hodnota je označena v tabulce 3 jako W. Modul FRDM-K64F tuto hodnotu předá ve formě délky pulzu servomotoru a ten natočí přední kola. Konstrukce použitého modelu automobilu zajišťuje splnění Ackermannovy podmínky.

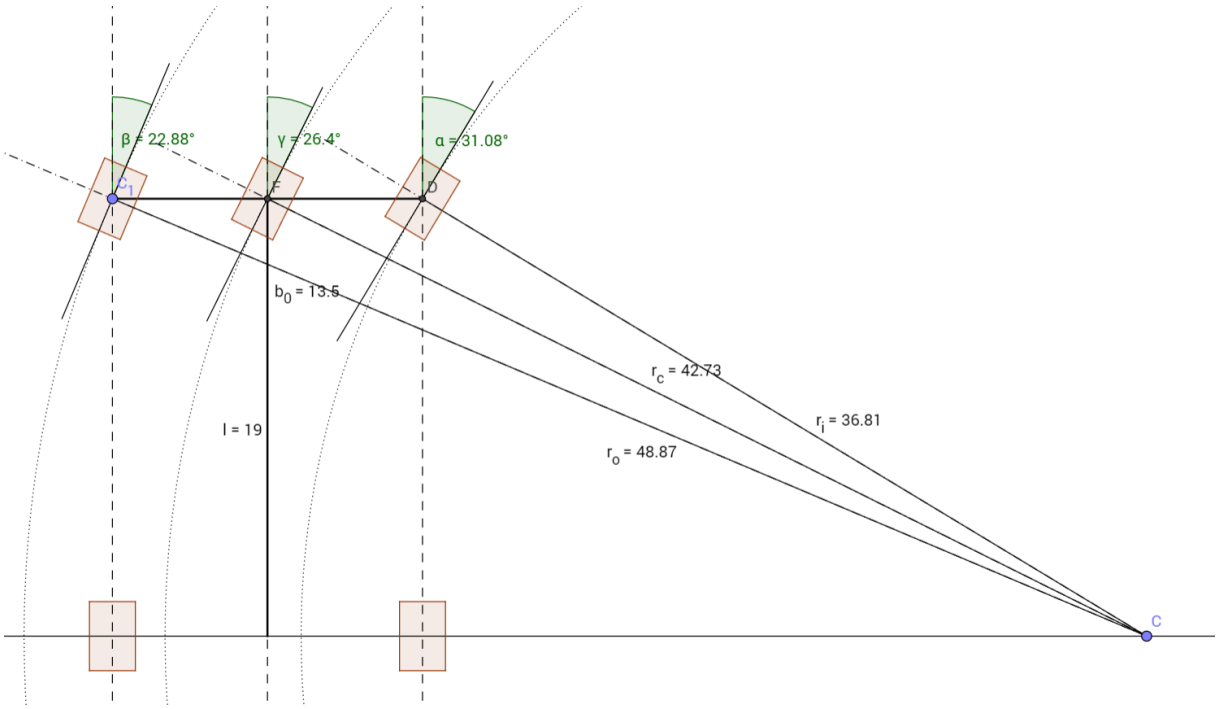
Zatáčení automobilu pomocí hodnot ±1000 není pro úlohu automatického parkování zcela vhodné. Výhodnější způsob určování směru automobilu je podle úhlu natočení pomyslného pro-

středního kola. Převod mezi úhlem natočení prostředního kola a odesílanou hodnotou pro ovládání servomotoru je dána výrazem (2):

$$f(x)\begin{cases} -1000, & x \leq \lambda_{-1} \\ -500 - \frac{x+|\lambda_{-0.5}|}{|\lambda_{-0,5}-\lambda_{-1}|} * 500 & \lambda_{-1} < x \leq \lambda_{-0,5} \\ -\frac{x}{|\lambda_{-0,5}|} * 500 & \lambda_{-0,5} < x \leq 0 \\ \frac{x}{|\lambda_{+0,5}|} * 500 & 0 < x \leq \lambda_{+0,5} \\ 500 + \frac{x-\lambda_{+0.5}}{\lambda_{+1}-\lambda_{+0,5}} * 500 & \lambda_{+0,5} < x \leq \lambda_{+1} \\ 1000 & \lambda_{+1} < x, \end{cases} \tag{2}$$

kde konstanta λ_{-1} odpovídá úhlu natočení prostředního kola při hodnotě -1000, $\lambda_{-0,5}$ odpovídá úhlu natočení prostředního kola při hodnotě W -500. Analogicky je tomu u ostatních konstant. Výraz (2) je v podstatě lineární interpolace provedená mezi šesti body.

Konstanty λ byly vypočteny z naměřených poloměrů otáčení použitého modelu automobilu. Poloměry otáčení byly naměřeny pro natočení kol při hodnotách W -1000, -500, 500 a 1000 a vždy vzhledem k vnitřnímu kolu. Z poloměrů byly pak vypočítány úhly natočení prostředního kola, tedy hodnoty konstant λ.



Obrázek 8: Ackermannova geometrie u použitého modelu auta

Pro měření poloměru otáčení byla kola natočena podle hodnoty, pro kterou bylo měření prováděno. Při tomto natočení kol byl automobil nastaven na konstantní malou rychlost a projížděl kružnicí danou požadovaným poloměrem. V této kružnici byly zaznamenávány vzdálenosti

Tabulka 3: Naměřené hodnoty modelu automobilu

Hodnota W	λ	α **(°)**	γ **(°)**	β **(°)**	r_i **(cm)**	r_c **(cm)**	r_o **(cm)**
-1000	λ_{-1}	-33,9	-28,5	24,5	-34,0	-39,8	-45,8
-500	$\lambda_{-0,5}$	-17,0	-15,4	-14,1	-64,9	-71,4	-77,9
500	$\lambda_{0,5}$	15,2	13,9	12,8	72,4	78,9	85,5
1000	λ_1	31,0	26,4	22,8	36,8	42,7	48,8

mezi trojicemi bodů. Zpravidla bylo výstupem jednoho měření několik trojic bodů, které tvoří trojúhelníky. Poloměr otáčení lze získat z kružnice opsané takovému trojúhelníku a vzhledem k tomu, že je těchto trojúhelníku více, je vzat aritmetický průměr vypočtených poloměrů. Naměřené hodnoty jsou v tabulce 3. Na obrázku 8 je ukázána Ackermannova geometrie u použitého modelu auta.

S těmito naměřenými hodnotami a převodním vztahem (2) lze nyní ovládat směr jízdy automobilu pomocí úhlu natočení prostředního kola. Pro řešení automatického parkování by bylo také vhodné mít k dispozici možnost ovládaní směru jízdy pomocí poloměru otáčení. Proto je použit převodní vztah mezi poloměrem otáčení prostředního kola a hodnotou přijímanou modulem FRDM-K64F, resp. převodní vztah (3) mezi poloměrem otáčení prostředního kola a úhlem natočení prostředního kola.

$$f(r_c) = 90° - \arccos\left(\frac{l}{r_c}\right) \tag{3}$$

Díky těmto naměřeným hodnotám a převodním vztahům lze určovat směr jízdy pomocí úhlu natočení pomyslného prostředního kola, nebo určením poloměru otáčení pomyslného prostředního kola.

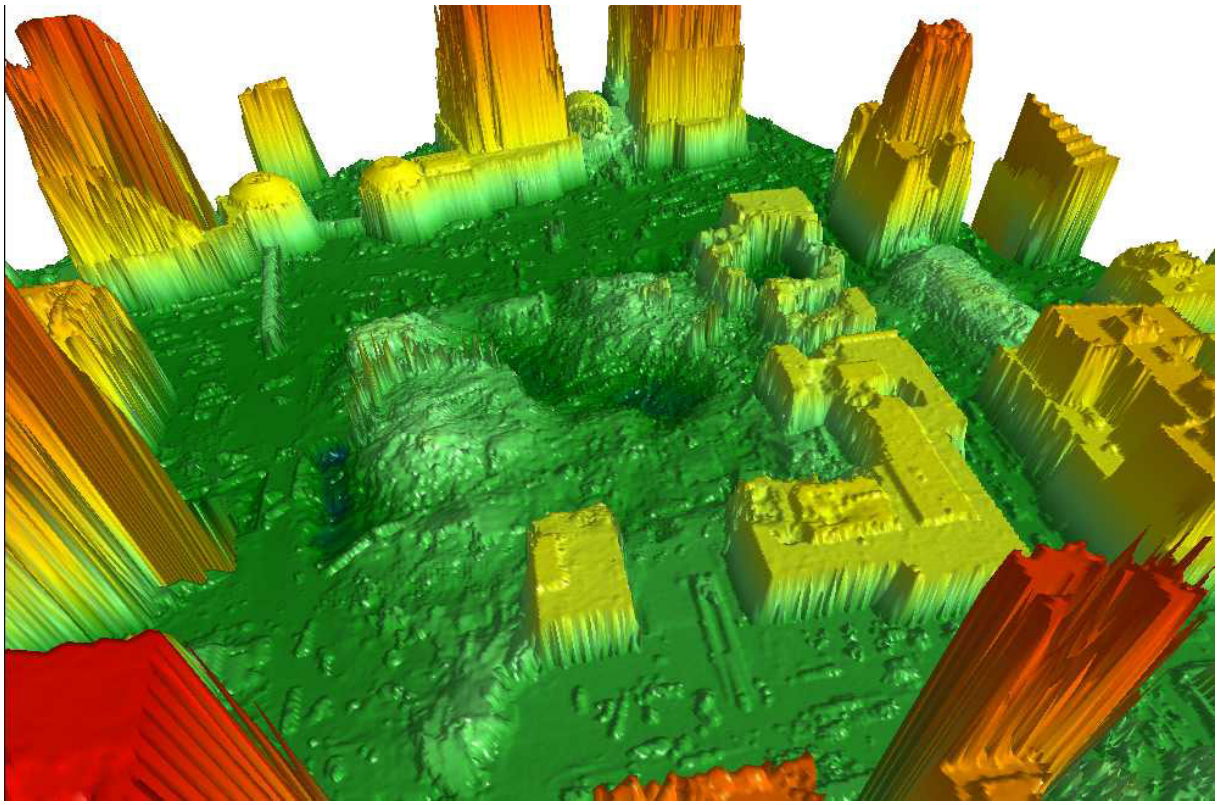
5 Laserový měřič vzdáleností (LRF)

Jedná se o zařízení, které využívá technologii LIDAR, což je metoda měření vzdáleností ke snímanému objektu, resp. překážce.

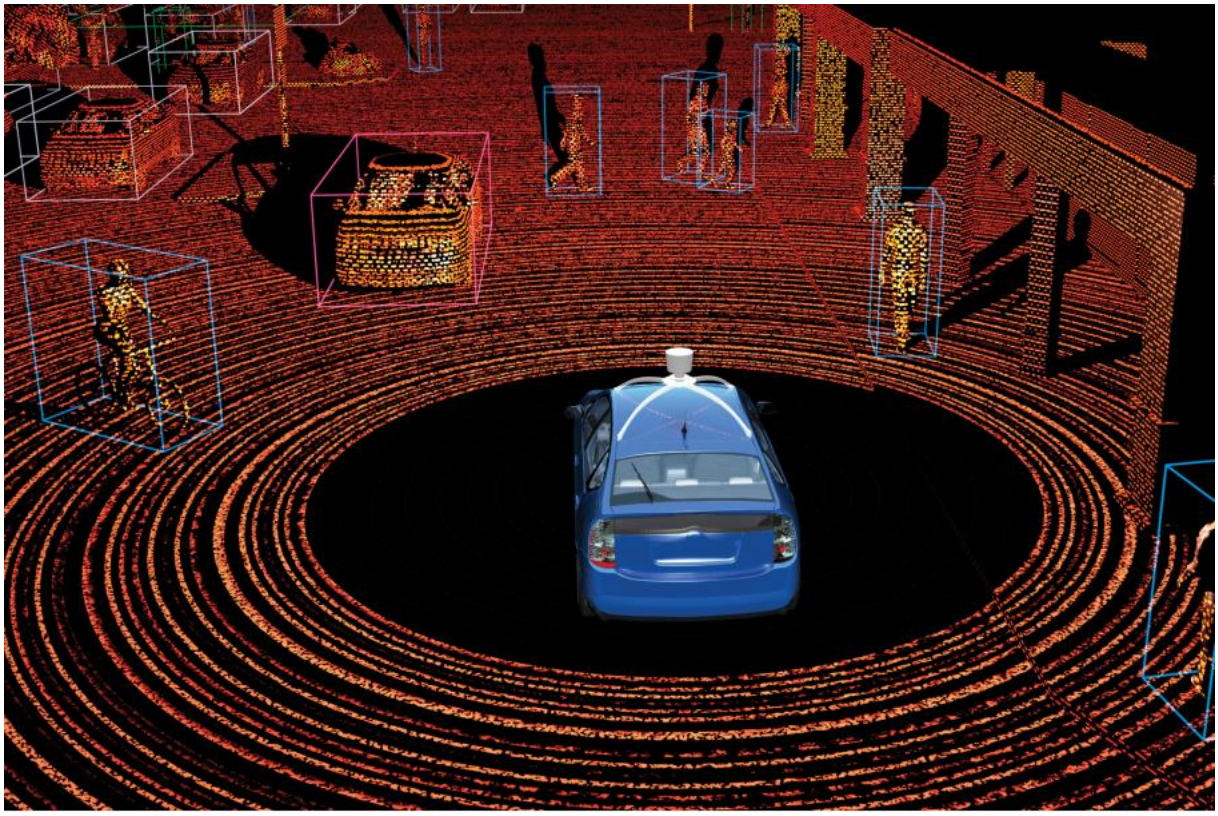
Používaná vlnová délka zavisí na způsobu použití skeneru. Pro nevědecké účely se používají vlnové délky paprsku mezi 600 až 1000 nm, a to z toho důvodu, že taková zařízení nejsou příliš drahá. Nicméně tato vlnová délka může být zachycena lidským okem, a proto laser nemůže vysílat na maximální výkon. Další varianta vlnové délky je 1550 nm, která je pro oko bezpečná i při vyšším výkonu. Nicméně problémem je fotodetektor, který není pro tuto vlnovou délku dostatečně přesný. Tato varianta je využívána hlavně ve vojenském průmyslu, protože není viditelná přes brýle pro noční vidění. Lasery pro vzdušnou topografii používají většinou vlnovou délku 1064 nm. Hloubkové systémy používají vlnovou délku 532 nm, protože proniká vodou s menším útlumem než 1064 nm, a tak je měření přesnější [5].

5.1 Využití technologie LIDAR

Technologie LIDAR se využívá v oblastech, kde je zapotřebí zmapovat určitou oblast, např. vytváření výškových map viz obrázek 9, důlní průzkum, orientace autonomních vozidel viz obrázek 10 [5].



Obrázek 9: Výšková mapa města New York pořízená snímáním technologií LIDAR



Obrázek 10: Snímek pořízený technologií LIDAR

5.1.1 Autonomní vozidla

Vozidla s autonomním ovládáním využivají skener LIDAR pro snímání svého okolí. Na základě mračna bodů vozidlo určuje směr jízdy tak, aby se orientovalo ve svém okolí. Při skenování se také buduje mapa okolí a rozpoznávají objekty [5].

5.1.2 Archeologie

Technologie LIDAR poskytuje archeologům možnost vytvářet digitální snímky s vysokým rozlišením archeologických míst, které mohou odhalit mikrotopografii, která je jinak skrytá vegetací [5].

5.1.3 Měření rychlosti

Policisté využívají radary pro měření rychlosti vozidla, které jsou založeny na technologii LIDAR [5].

5.1.4 Fyzika a astronomie

LIDAR je využíván astronomy k měření vzdáleností od odrážek umístěných na měsíci. Díky tomuto měření dokáží určit polohu měsíce s přesností na několik milimetrů [5].

5.2 Princip laserového skeneru

Laserový skener vzdáleností je složen z několika částí:

1. Laser, který vysílá světelné paprsky o určité vlnové délce a pod úhlem, který se v průběhu skenování mění. Jak se úhel během skenování mění závisí na typu skeneru. U 2D skenerů se úhel paprsku mění kolem jedné osy, a proto výsledné vzdálenosti od snímaného objektu ke skeneru tvoří rovinu. U 3D skenerů se úhel paprsku mění podle dvou os a vzdálenosti pak tvoří mračno bodů, které reprezentuje naskenovaný 3D prostor.

2. Optická část, která obstárává nasměrování vyzařovaného paprsku. Může se jednat o zrcátko připevněné ke krokovému motoru. Elektronika skeneru nastaví polohu zrcátka na požadovaný úhel, laserový paprsek se vyšle do zrcátka.

3. Fotodetektor, který přijímá zpětný rozptyl, na jehož základě se určí vzdálenost od snímaného objektu.

Laser tedy vyšle pulz, který je směrován pomocí optické části, fotodetektor vyslaný pulz přijme a vyhodnotí informaci o vzdálenosti od snímaného objektu. Toto snímaní probíhá opět s tím rozdílem, že optická část zařídí nasměrování paprsku o větší úhel. Tento proces se opakuje, dokud není nasnímáno celé okolí [6].

Pro výpočet vzdálenosti k snímanému objektu je potřeba změřit čas, který uplynul mezi vysláním paprsku až do doby, kdy fotodetektor přijme zpětný rozptyl paprsku. Vzhledem k vysoké rychlosti světla není tato metoda vhodná pro přesné měření vzdáleností. Lze také měřit rychlost snímaného objektu vůči skeneru, a to pomocí Dopplerova jevu.

Vzdálenost mezi skenerem a snímaným objektem je dána vztahem (4):

$$D = \frac{ct}{2}, \tag{4}$$

kde c je rychlost světla v atmosféře a t je změřená doba od vyslání paprsku do přijetí zpětného rozptylu. Doba t je dána vztahem (5):

$$t = \frac{\varphi}{\omega}, \tag{5}$$

kde φ je fázové zpoždění a ω je úhlová frekvence. Po dosazení je vzdálenost popsána vztahem (6):

$$D = \frac{1}{2}ct = \frac{c\varphi}{2\omega} = \frac{c}{4\pi f}(N\pi + \Delta\varphi) = \frac{\lambda}{4}(N + \Delta N), \tag{6}$$

kde λ je vlnová délka paprsku ($\frac{c}{f}$), $\Delta\varphi$ je část fázového zpoždění (φ *modulo* π), N je počet půlvln pro cestu mezi skenerem a snímaným objektem a ΔN je zbývající část půlvln [6].

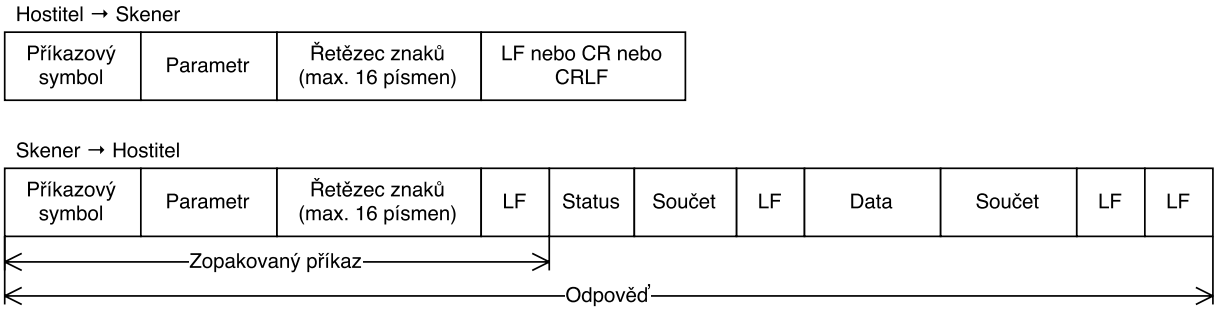
5.3 Skener Hokuyo URG-04LX

V této práci byl využit skener Hokuyo URG-04LX viz obrázek 11. Skener je napájen 5 V ze stejnosměrného měniče napětí a připojen k hlavnímu počítači prostřednictvím USB portu viz obrázek 2. Napájení pro skener nemohlo být použito z USB portu, a to vzhledem k velkému krátkodobému proudovému zatížení, vzniklém při roztočení motoru ve skeneru. Skener používá laser s vlnovou délkou 785 nm. Jeho limity změřitelné vzdálenosti jsou od 20 mm do 4000 mm ±3%. Rozsah skeneru je 240° a krok posunu je 0,36°. Skener tedy ukládá okolní vzdálenosti do 683 hodnot [2].

Obrázek 11: Hokuyo URG04-LX skener

5.3.1 Komunikace se skenerem

Skener komunikuje po virtuální sériové lince. Komunikace probíhá tak, že hlavní počítač vždy pošle příkaz skeneru a ten mu odpoví. Struktura datového rámce je zobrazena na obrázku 12.



Obrázek 12: Struktura datového rámce komunikace s Hokuyo URG00-LX skenerem

Příkazový symbol je 2 bytový kód na začátku každého datového rámce. Každý příkaz má svůj specifický symbol. Parametr předává informace potřebné pro změnu nastavení skeneru nebo

pro vyžádání dodatečných údajů. Řetězec znaků je volitelná informace v příkazu a používá se k ověření odpovědi.

Status je 2 bytový kód, který reprezentuje stavový kód operace. Data jsou každých 64 bajtů oddělena `LF` (`0x0A`).

Ve specifikaci komunikace existuje 13 příkazů, které ovládají skener. Například příkaz `CR` určuje rychlost motoru. Tato změna rychlosti je vyžadována v prostředí, kde je použit více než jeden skener, aby se zabránilo vzájemnému rušení paprsků. Příkaz `VV` získá detaily o skeneru: verze firmware, sériové číslo, atd.

Příkaz `GD` nebo `GS` slouží pro získání posledního snímku naměřených hodnot. Data jsou kódována ve třech možných variantách.

První je dvou znakové kódování, při kterém se přenáší vzdálenost zakódována do 8 bitů. Hodnota v milimetrech je rozdělena do dvou 6 bitových hodnot, ke každé je přičteno 30_{16} a následně jsou tyto hodnoty převedeny na znaky podle ASCII tabulky. Například hodnota 1234 mm je převedena na dvě 6 bitové hodnoty 010011_2 a 010010_2, následně je ke každé přičtena hodnota 30_{16}, tudíž budou hodnoty 43_{16} a 42_{16} a v posledním kroku jsou hodnoty převedeny na znaky C a B. Výsledná zakódována hodnota je CB. Pro převod zakódované hodnoty zpět se postupuje obráceně. Zakódovaný řetězec CB se převede na hexadecimální hodnoty 43_{16} a 42_{16} podle ASCII tabulky. Následně je od těchto hodnot odečteno 30_{16}. Po tomto kroku budou hodnoty 13_{16} a 12_{16}. Nyní se spojí ve své binární podobě na hodnotu 010011010010_2 a to odpovídá vzdálenosti 1234 mm.

Další dvě kódování jsou tří a čtyř znakové, u kterých je princip stejný jako u dvou znakového kódování, jen jsou vzdálenosti rozděleny do tří, resp. čtyř 6 bitových hodnot [8].

6 Simultánní lokalizace a mapování

U úloh, které vyžadují, aby se robot orientoval, resp. autonomně pohyboval v prostoru, se používá SLAM. Robot si vytváří informaci o svém okolí a poloze. Mapa okolí se generuje neustále a při pohybu robota se rozšiřuje o části, které skener ještě nebyl schopen naskenovat.

SLAM je problém sestavování a aktualizace mapy neznámého prostředí, zatímco se sleduje poloha robota. Pro řešení SLAM bylo vytvořeno mnoho aproximačních algoritmů. Některé vyžadují prezenci klíčových bodů, např. rohů v případě vnitřního užití. U algoritmů SLAM se neklade větší důraz na přesnost, ale často jsou přizpůsobovány dostupným výpočetním zdrojům [7].

6.1 ICP-SLAM

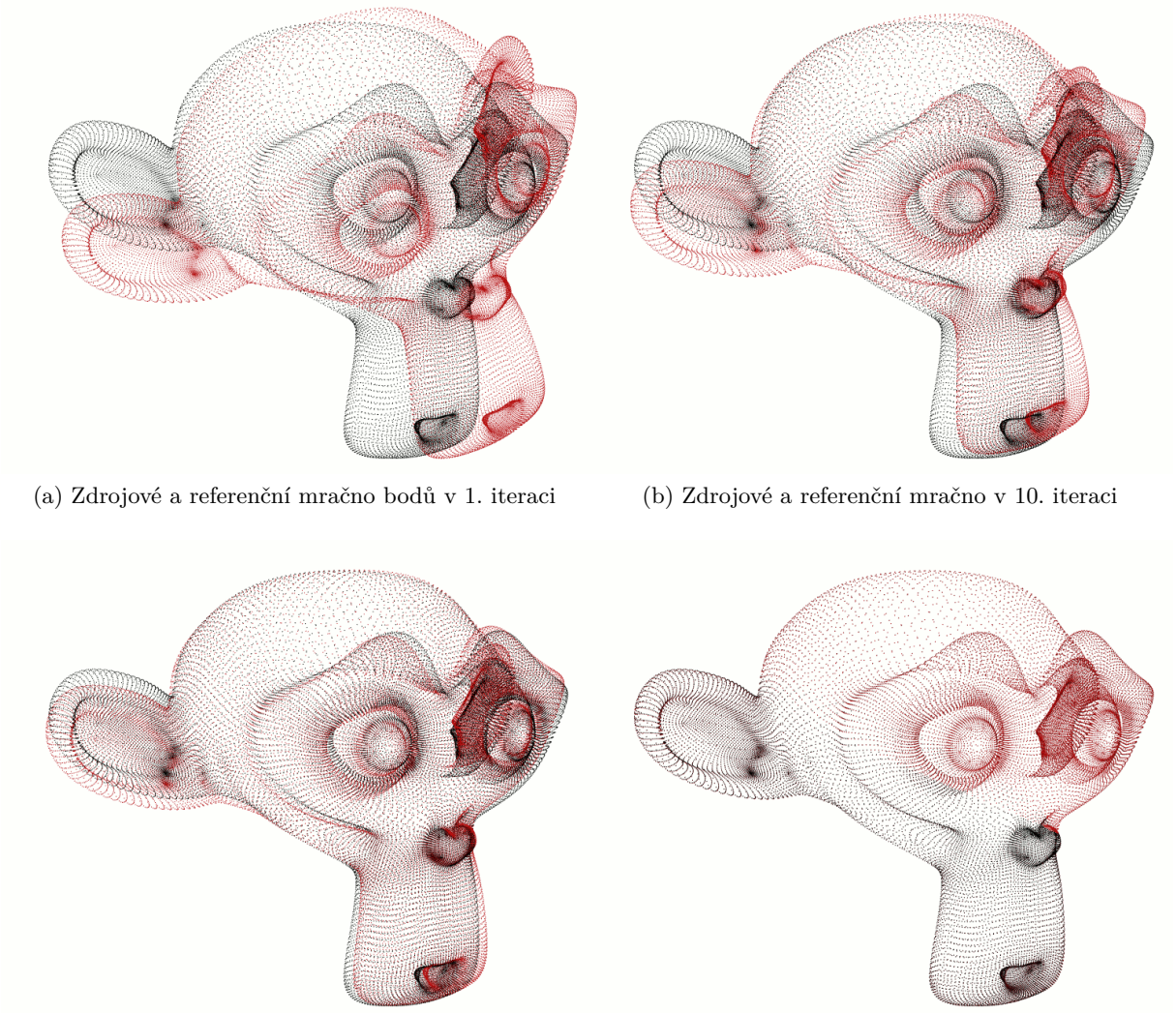
Jedná se o řešení SLAM pomocí algoritmu Iterative Closest Point (ICP). Algoritmus ICP hledá translačně-rotační transformace minimalizující vzdálenosti mezi dvěmi mračny bodů. Ukázka je na obrázku 13. Kroky algoritmu jsou:

1. Pro každý bod ve zdrojovém mračnu najdi nejbližší bod v referenčním mračnu (nebo ve vybrané množině bodů z mračna).
2. Odhadni translačně-rotační transformaci pomocí techniky minimalizace vzdáleností od středů dvou mračen (nebo množin bodů).
3. Aplikuj transformaci na zdrojové mračno.
4. Opakuj od bodu 1 až do vyčerpání předem určeného počtu iterací, nebo do chvíle kdy bude chyba dostatečně malá.

U řešení SLAM pomocí ICP odpovídá zdrojové mračno bodů novému snímku prostředí a referenční mračno odpovídá mapě prostředí. Do mapy prostředí se vždy přidává správně tranformovaný snímek. Tímto způsobem je zajištěna aktualizace mapy a lokalizace robota. Takto generované mapě neustále přibývá počet bodů, což vede k vyšší paměťové a výpočetní náročnosti.

6.2 Reprezentace 2D mapy pomocí mřížky obsazenosti

Mapa může být také reprezentována mřížkou, čímž by se snížila paměťová složitost u delšího běhu ICP-SLAM algoritmu. U většího počtu bodů v mapě je mřížka obsazenosti z hlediska paměťové úspory vhodnější reprezentace mapy. Buňky mřížky mají pevně stanovenou velikost a při vkládání bodu se body, které jsou blízko sebe, sloučí do stejné buňky. Buňky mřížky drží informaci o tom, zda se na daném místě nachází překážka, volný prostor, nebo místo ještě nebylo prozkoumáno. V podstatě se jedná o číselnou hodnotu, která reprezentuje pravděpodobnost překážky nebo volného prostoru s tím, že prvotní hodnoty buněk drží hodnotu, která reprezentuje, že dané místo ještě nebylo naskenováno. Po nějaké době vkládání bodů z 2D skeneru do mapy



(a) Zdrojové a referenční mračno bodů v 1. iteraci

(b) Zdrojové a referenční mračno v 10. iteraci

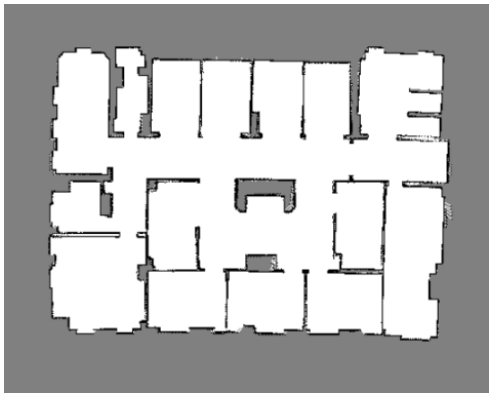
(c) Zdrojové a referenční mračno v 15. iteraci

(d) Zdrojové a referenční mračno ve 20. iteraci

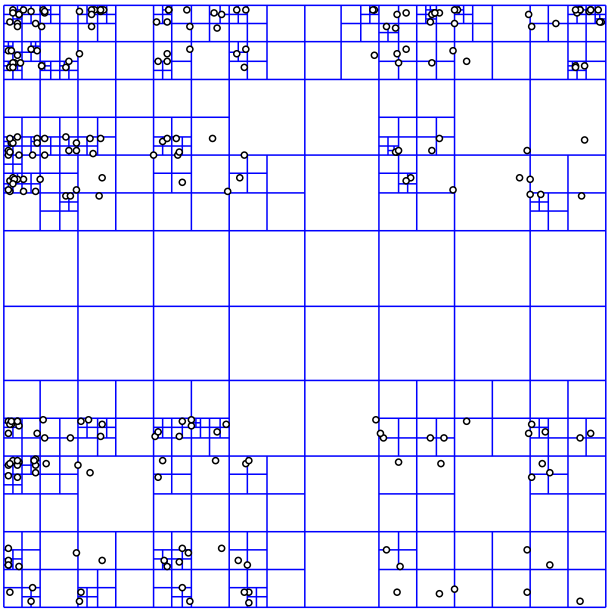
Obrázek 13: Ukázka algoritmu ICP (černě referenční mračno, červeně zdrojové)

pomocí ICP-SLAM algoritmu a pohybu robota, může vypadat výsledná dvourozměrná mřížka obsazenosti jako na obrázku 14, kde bílá barva reprezentuje volný prostor, černá barva překážku a šedá barva dosud neprozkoumaná místa. Velikost buňky mřížky na obrázku 14 je 10 cm. Velikost buňky určuje rozlišení mřížky a určení této velikosti závisí na požadované aplikaci.

Mřížka obsazenosti využívá datovou strukturu kvadrantový strom pro ukládání bodů. Ukázka na obrázku 15. V každém listu stromu je informace o buňce mřížky. Díky vlastnostem kvadrantového stromu lze mřížku rozšiřovat v libovolném směru. Časová složitost pro vyhledání buňky ve stromu je $\mathcal{O}(h)$, kde h je hloubka stromu. U úplného kvadrantového stromu s n listy je hloubka rovna $h = \log_4 n$, výsledná složitost vyhledání buňky je tedy $\mathcal{O}(\log_4 n)$.



Obrázek 14: Příklad mřížky obsazenosti po projetí budovou s robotem [1]



Obrázek 15: Vizualizace kvadrantového stromu s několika vloženými body [2]

Při vkládání transformovaného mračna bodů do mřížky se vždy upraví buňky, kterými prochází paprsek a buňka (resp. buňky), která je v oblasti odražení paprsku. V místě odražení paprsku se v buňce (resp. buňkách) zvýší pravděpodobnost výskytu překážky a v buňkách, kterými paprsek prochází se zvýší pravděpodobnost výskytu volného prostoru.

ICP-SLAM může využívat jako referenční mapu mřížku obsazenosti a následně správně transformované mračna bodů vždy přidávat do mřížky, čímž se mřížka aktualizuje pohybem robota. V této práci je využit ICP-SLAM algoritmus a jako reprezentace mapy je využita mřížka obsazenosti. Velikost buňky v mřížce obsazenosti je 5 mm, vzhledem k rozměrům modelu automobilu

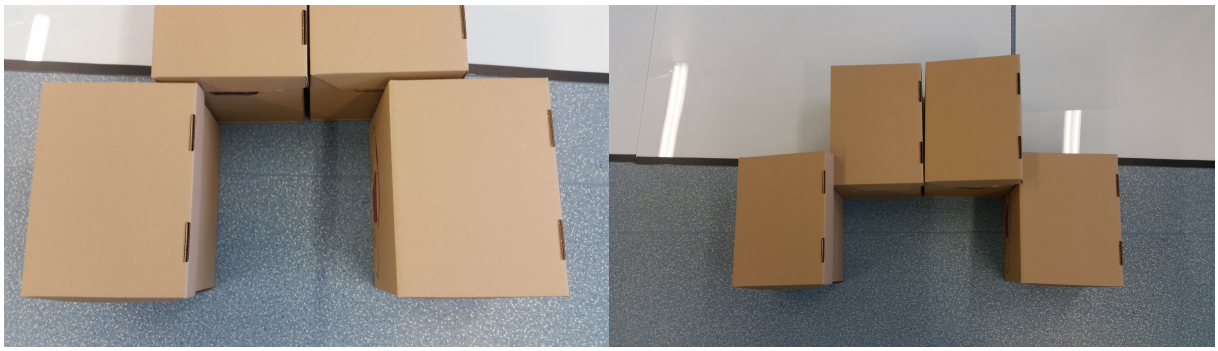
[1] robotang. [online]. [cit. 2017-04-15]. Dostupné z: `http://robotang.co.nz/projects/robotics/custom-player-plugins/`

[2] Quadtree. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2017-04-15]. Dostupné z: `https://en.wikipedia.org/wiki/Quadtree`

a úlohy automatického parkování.

7 Detekce parkovacího místa

V této práci je parkovací místo uměle vytvořený prostor o rozměrech, které odpovídají modelu automobilu. Na obrázku 16 jsou vyfoceny uměle vytvořené parkovací místa. Pro vytvoření překážek je v této práci použito lepenkových krabic. Tyto krabice jsou poskládany tak, aby vytvořily prostor, který připomíná parkovací místo pro podélné a kolmé parkování.



(a) Ppro kolmé parkování (b) Pro podélné parkování

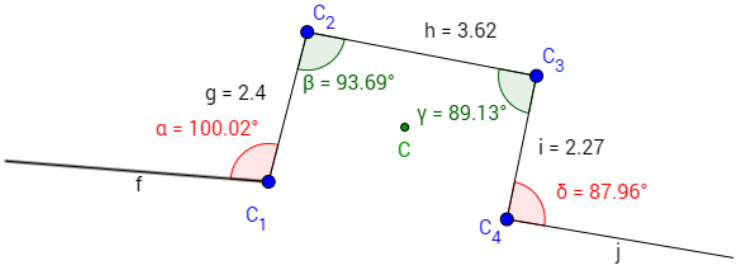
Obrázek 16: Uměle vytvořená parkovací místa

Parkovací místo je v této práci určeno čtyřmi rohovými body. Na obrázku 17 je ukázkové parkovací místo pro podélné parkování a rohové body jsou C_1, C_2, C_3 a C_4. Z těchto bodů je následně vypočítán střed parkovacího místa, který je na obrázku vyznačen jako bod C. Ze vzdáleností mezi těmito body je určen typ parkovacího místa. Typy parkovacích míst mohou být:

1. Parkovací místo vhodné pouze pro podélné parkování.
2. Parkovací místo vhodné pouze pro kolmé parkování.
3. Parkovací místo vhodné pro oba způsoby parkování.
4. Nevyhovující parkovací místo.

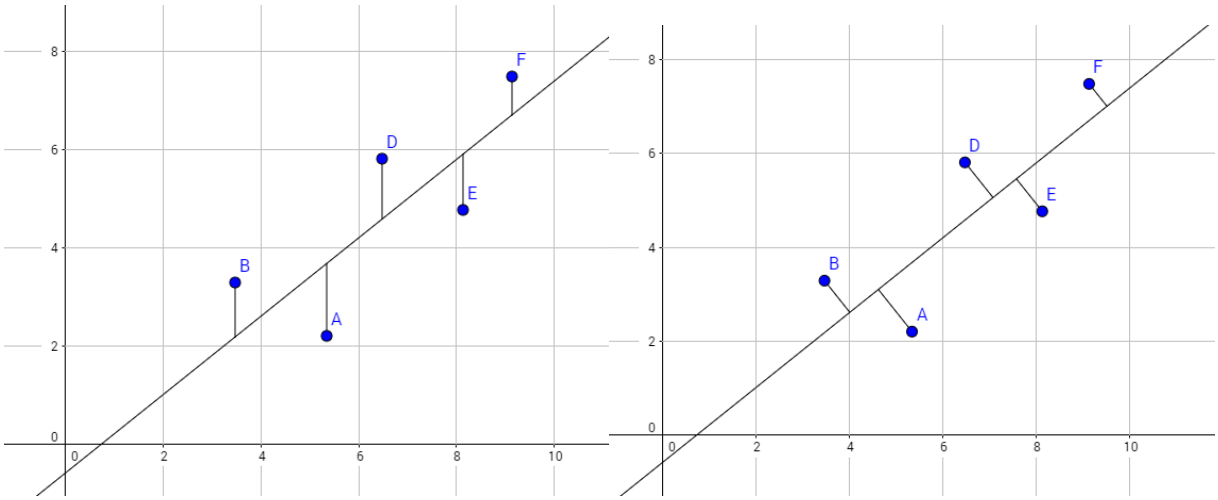
7.1 Určení přímky proložené množinou bodů

Aby bylo možné detekovat parkovací místo, je zapotřebí získat z mračna bodů přímky (resp. úsečky), které co nejvíce kopírují dané mračno bodů. Lineární regrese je matematická metoda používaná pro proložení souboru bodů přímkou. Metoda nejmenších čtverců je nejjednodušší a nejčastěji používaná forma lineární regrese a poskytuje řešení problému hledání přímky, která prochází souborem bodů s nejmenší odchylkou, resp. s nejmenším kvadrátem vzdáleností bodů od přímky [9].



Obrázek 17: Příklad parkovacího místa pro podélné parkování

Vzhledem k tomu, že v řešené úloze mohou vznikat přímky obecného směru, je chápána vzdálenost mezi bodem a generovanou přímkou jako délka úsečky tvořená kolmicí ke generované přímce, která prochází bodem viz obrázek 18.



(a) Vertikální vzdálenosti

(b) Kolmé vzdálenosti

Obrázek 18: Rozdíl mezi kolmými a vertikálními vzdálenostmi k přímce

Součet kolmých vzdáleností od bodu k vytvářené přímce je definována vztahem 7:

$$R_{\perp} = \sum_{i=1}^{n} d_i, \tag{7}$$

kolmá vzdálenost bodu k přímce je definován vztahem: 8:

$$d_i = \frac{|y_i - (a + bx_i)|}{\sqrt{1 + b^2}}, \tag{8}$$

funkce, která má být minimalizována je dána vztahem 9:

$$R_{\perp} = \sum_{i=1}^{n} \frac{|y_i - (a + bx_i)|}{\sqrt{1 + b^2}}. \tag{9}$$

$$R_{\perp}^2 = \sum_{i=1}^{n} \frac{[y_i - (a + bx_i)]^2}{1 + b^2} \tag{10}$$

Bohužel funkce absolutní hodnoty nemá spojitou derivaci, tudíž není minimalizace výrazu $R_{\perp}$ vhodná pro analytické řešení. Nicméně pokud je kvadrát vzdáleností dle výrazu 10 minimalizován, může být problém vyřešen analyticky výrazy 11 a 12:

$$\frac{\partial R_{\perp}^2}{\partial a} = \frac{2}{1 + b^2} \sum_{i=1}^{n} [y_i - (a + bx_i)](-1) = 0, \tag{11}$$

$$\frac{\partial R_{\perp}^2}{\partial b} = \frac{2}{1 + b^2} \sum_{i=1}^{n} [y_i - (a + bx_i)](-x_i) + \sum_{i=1}^{n} \frac{[y_i - (a + bx_i)]^2(-1)(2b)}{(1 + b^2)^2} = 0. \tag{12}$$

Po úpravách rovnic 11 a 12 se lze dostat ke kvadratické rovnici 13 a rovnici 14:

$$b^2 + \frac{\sum_{i=1}^{n} y_i^2 - \sum_{i=1}^{n} x_i^2 + \frac{1}{n}[(\sum_{i=1}^{n} x_i)^2 - (\sum_{i=1}^{n} y_i)^2]}{\frac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i y_i} b - 1 = 0, \tag{13}$$

$$a = \frac{\sum_{i=1}^{n} y_i - b \sum_{i=1}^{n} x_i}{n}, \tag{14}$$

výraz 13 lze řešit rovnicí 15:

$$b = -B \pm \sqrt{B^2 + 1}, \tag{15}$$

kde B je vyjádřeno výrazem 16:

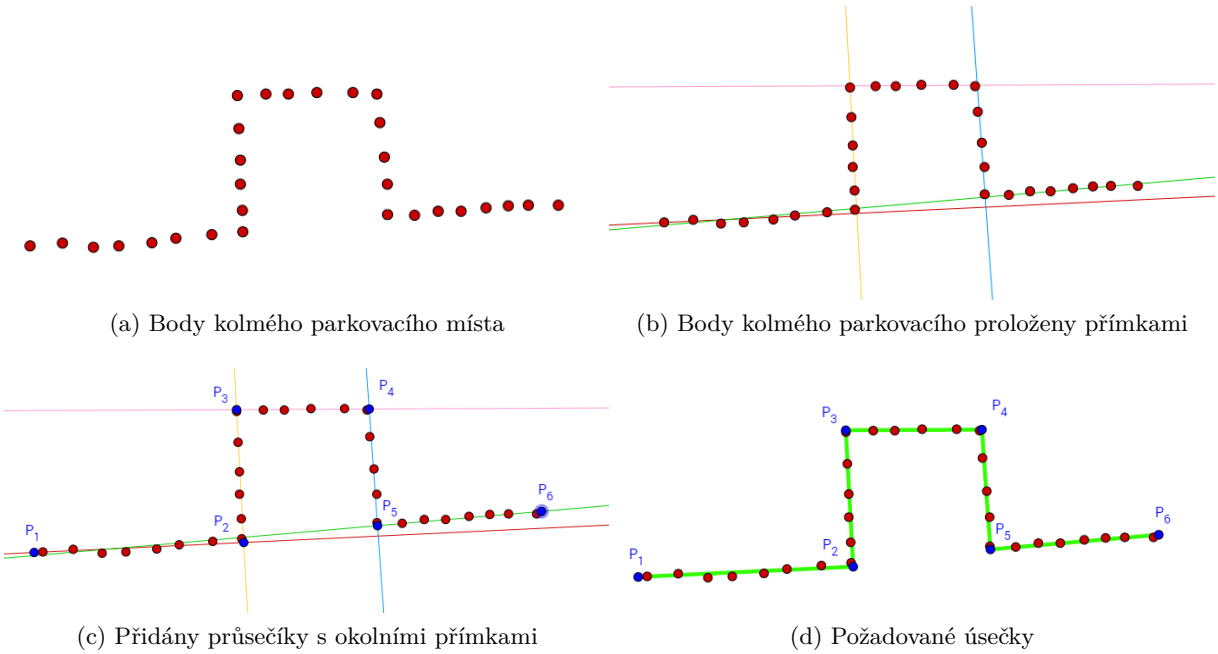
$$B = \frac{1}{2} \frac{[\sum_{i=1}^{n} y_i^2 - \frac{1}{n}(\sum_{i=1}^{n} y_i)^2] - [\sum_{i=1}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2]}{\frac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i y_i}. \tag{16}$$

Z řešení kvadratických rovnic je zvoleno to, které má menší chybu, resp. součet kvadrátů vzdáleností [9].

7.2 Převod mračna bodů na množinu úseček

Pro nalezení parkovacího místa slouží úsečky, které jsou proloženy mračnem bodů získaného ze skeneru viz obrázek 19. Body jsou proloženy přímkami pomocí výše zmíněné metody nejmenších čtverců. Z těchto přímek a jejich společných průsečíků jsou vytvořeny úsečky.

Algoritmus vytvoření přímek je vyjádřen pomocí diagramu aktivit na obrázku 20 a jeho implementace je ve výpise 2.



(a) Body kolmého parkovacího místa

(b) Body kolmého parkovacího proloženy přímkami

(c) Přidány průsečíky s okolními přímkami

(d) Požadované úsečky

Obrázek 19: Postup získávání úseček z bodů

V algoritmu hraje roli parametr, který určuje hranici pro ukončení přidávání bodů k přímce. Na obrázku 20 tento parametr figuruje v podmínce „Je chyba přímky malá?“. Pokud je parametr malý, je přímek generováno více a zachycují více detailů. U malých hodnot tohoto parametru je přímek méně a snižují počet zachycených detailů. Na obrázcích 21 je zobrazen rozdíl mezi vysokou a nízkou hodnotou parametru.

Z kolekce se nyní vytvoří průsečíky dvou sousedních přímek a těmito průsečíky se vždy korespondující přímky omezí, čímž se získají úsečky viz obrázek 19.

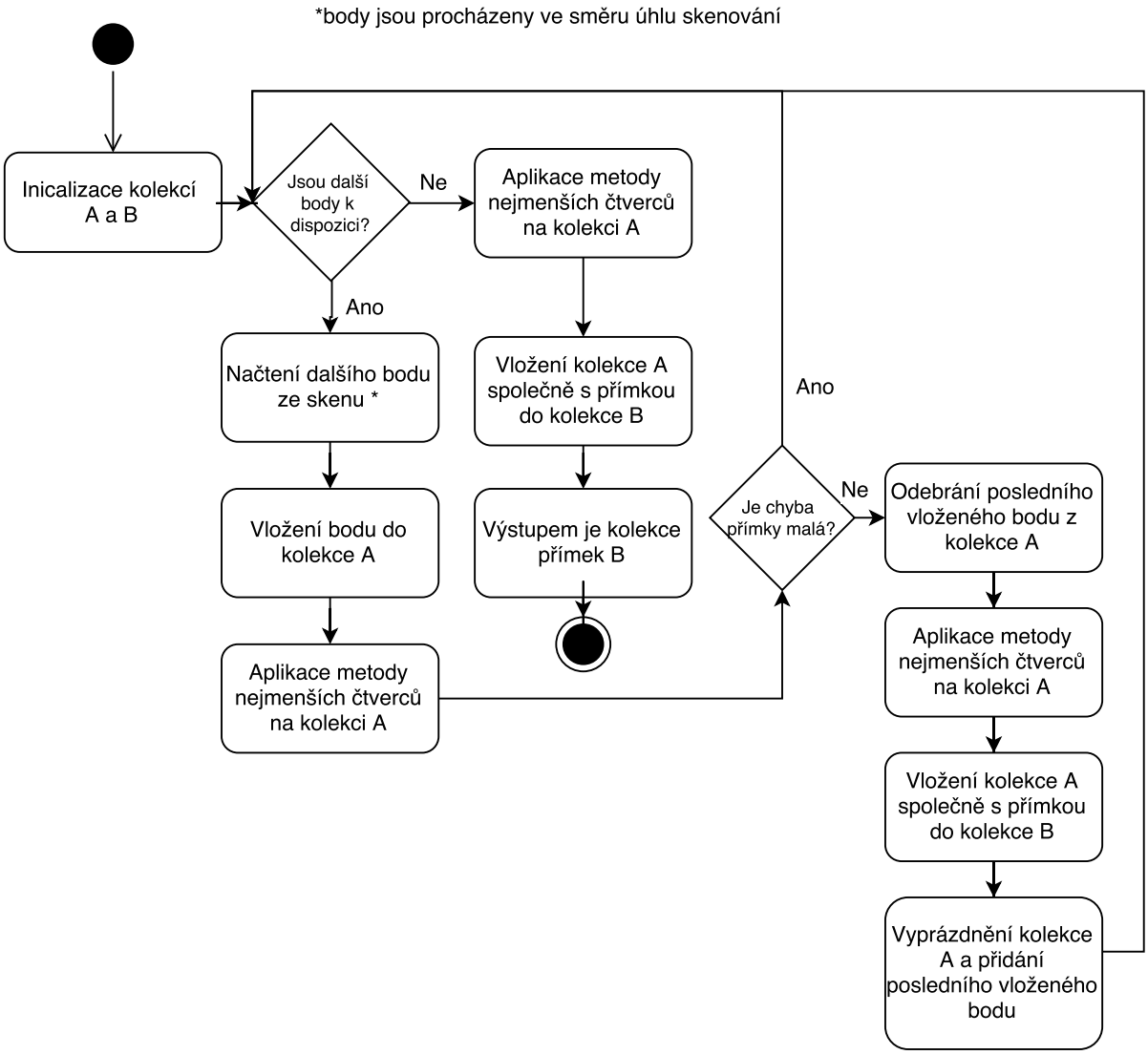
Takto vytvořená kolekce úseček může obsahovat úsečky, které jsou krátké viz úsečka q na obrázku 22. Takovéto úsečky jsou nežádoucí. Na obrázku 22 z parkovacího místa zdeformují roh, což ovlivní rozpoznávání místa.

Tyto nežádoucí úsečky se z kolekce odstraní a jejich sousední úsečky se prodlouží do průsečíků, jako je tomu na obrázku 23.

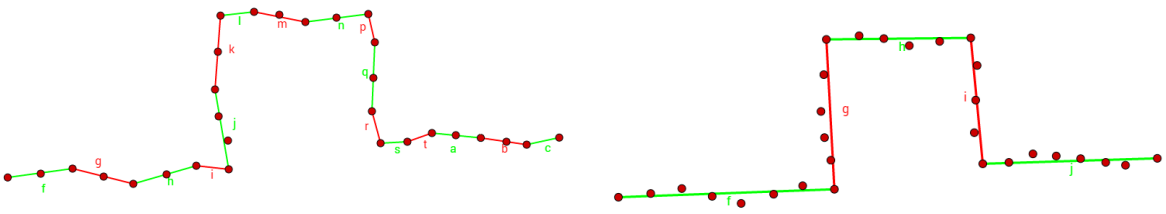
Úsečky tedy vymezují hranice mezi volným prostorem a překážkou. Nicméně v případě, kdy skener nezměří hodnotu vzdálenosti, kvůli svého limitu, jsou špatně vymezeny hranice v místech, kam skener „nedosáhne“. Na obrázku 24(a) je nežádoucí úsečka označena k. Jedná se tedy o úsečku, která je proložená nízkým množství bodů. Takovéto úsečky se z kolekce odstraní viz obrázek 24(b).

7.3 Algoritmus pro detekci parkovacího místa

Datová struktura popisující parkovací místo obsahuje informace o pozicích rohových bodů $C_{1...4}$, ze kterých se určí střed parkovacího místa, typ a další potřebné informace.



Obrázek 20: Diagram aktivit algoritmu pro generování přímek z mračna bodů

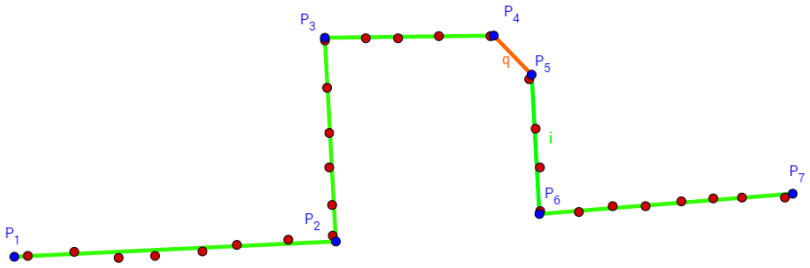


(a) Parametr je nastaven na nízkou hodnoty

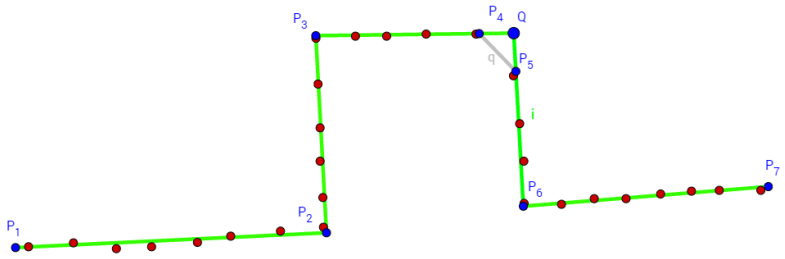
(b) Parametr je nastaven na vyšší hodnotu

Obrázek 21: Rozdíl mezi nízkou a vysokou hodnotou parametrem určující hranici chyby

Algoritmus detekce parkovacího místa v podstatě hledá čtveřici průsečíků sousedních přímek, svírajících přibližně pravý úhel. Ke změřenému úhlu je také přidána informace o tom, zda je proti



Obrázek 22: Ukázka úsečky q, která je tvořena pouze dvěmi body



Obrázek 23: Po odstranění nežádoucí úsečky

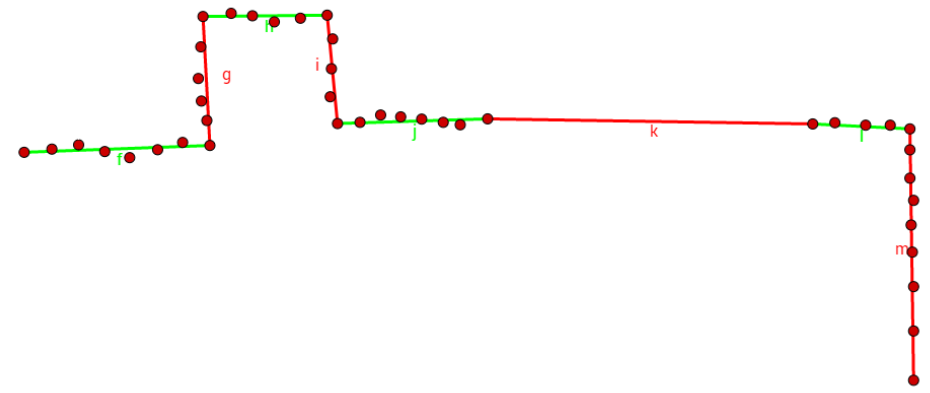
nebo ve směru hodinových ručiček, resp. je-li úhel kladný nebo záporný. S informací o směru úhlu lze nyní rozhodnout, zda je vedlejší úsečka směrována doprava (resp. doleva). Rohové úhly parkovacího místa dodržují vzor: 1× ve směru, 2× proti směru a 1× ve směru hodinových ručiček. Také by se dalo říci, že úhly dodržují vzor: 1× kladný, 2× záporný, 1× kladný [14].

Vzhledem k tomu, že body jsou seřazeny podle úhlu ve směru skenování, jsou i úsečky vytvořené pomocí metody nejmenší čtverců (popsáno v předešlé kapitole) seřazeny. V rámci detekce parkovacího místa se prochází kolekce seřazených úseček ve směru skenování. Vždy se vyberou 4 následující úsečky. Pokud jsou mezery mezi úsečkami větší než stanovená hranice, je testovaná skupina úseček automaticky vyřazena. Nyní se určí úhly mezi dvojicemi úseček. Pokud splňují výše zmíněný vzor, jsou tyto 4 úsečky označeny jako možné parkovací místo. Tento proces se opakuje s tím, že se druhá úsečka stane první až do chvíle, kdy jsou všechny úsečky vyčerpány. Diagram aktivit algoritmu pro detekci parkovacího místa je na obrázku 25 a část kódu je ve výpise 3.

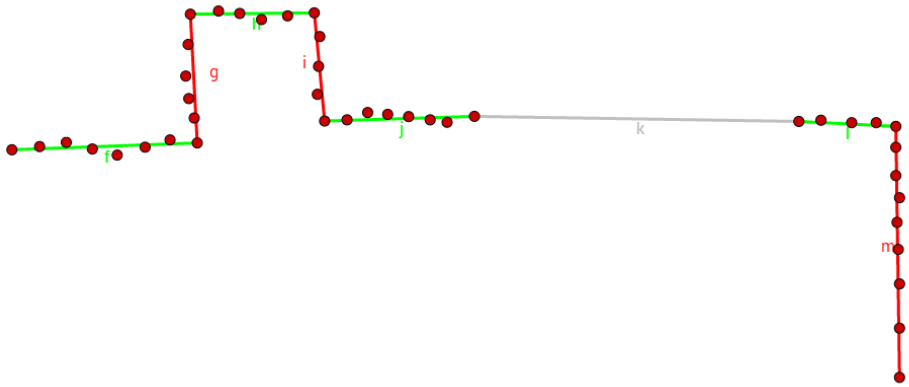
Po nalezení možných parkovacích míst je nyní potřeba určit středy míst, velikosti parkovacích míst a vyhodnotit jejich typy. Souřadnice středu C parkovacího místa určuje vztah 17:

$$C = \left(\frac{\sum_{i=0}^{4} C_{i_x}}{4}, \frac{\sum_{i=0}^{4} C_{i_y}}{4} \right), \tag{17}$$

kde body C_i jsou rohové body. Šířka W a délka H parkovacího místa jsou určeny vzdálenostmi mezi body $|C_1C_4|$ (také $|C_2C_3|$) a $|C_1C_2|$ (také $|C_4C_3|$). Rozměry jsou dány vztahy 18 a



(a) Nežádoucí úsečka k



(b) Nežádoucí úsečka k odstraněna

Obrázek 24: Před a po odstranění nežádoucí úsečky

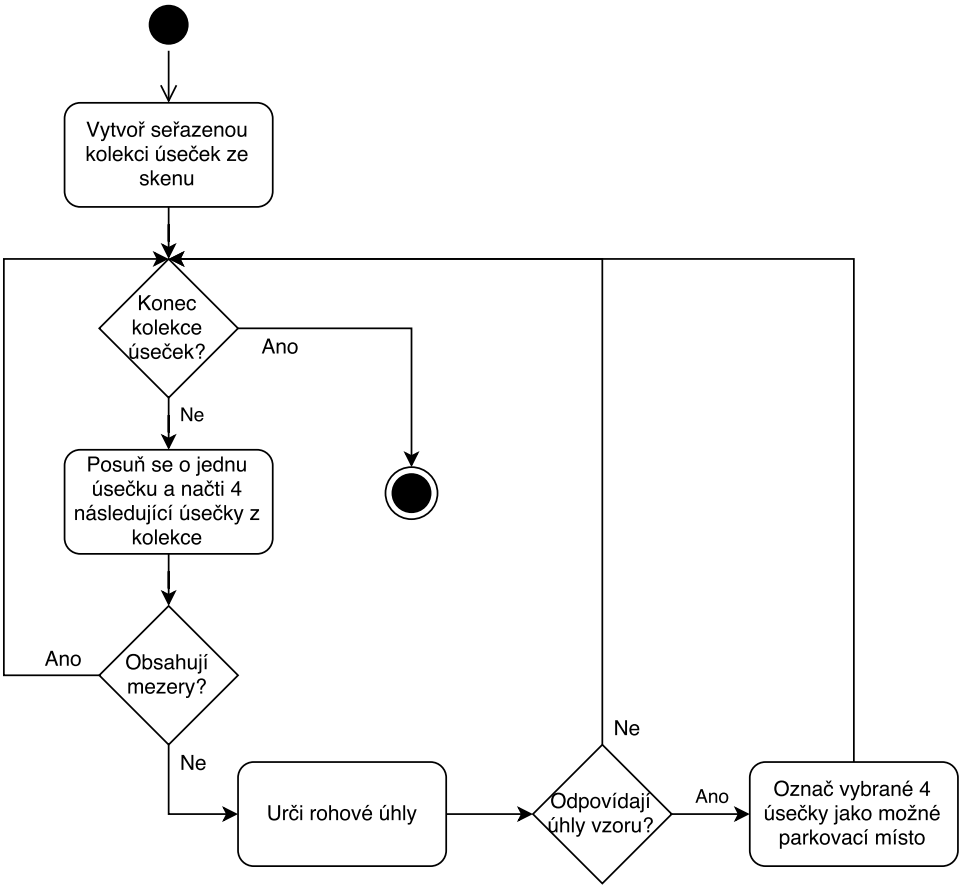
19:

$$H = \frac{|C_1C_4| + |C_2C_3|}{2} = \frac{\sqrt{(C_{4_x} - C_{1_x})^2 + (C_{4_y} - C_{1_y})^2} + \sqrt{(C_{3_x} - C_{2_x})^2 + (C_{3_y} - C_{2_y})^2}}{2}, \tag{18}$$

$$W = \frac{|C_1C_2| + |C_4C_3|}{2} = \frac{\sqrt{(C_{2_x} - C_{1_x})^2 + (C_{2_y} - C_{1_y})^2} + \sqrt{(C_{4_x} - C_{3_x})^2 + (C_{4_y} - C_{3_y})^2}}{2}. \tag{19}$$

Z rozměrů místa lze nyní určit, zda je parkovací místo vhodné pro podélné nebo kolmé parkování.

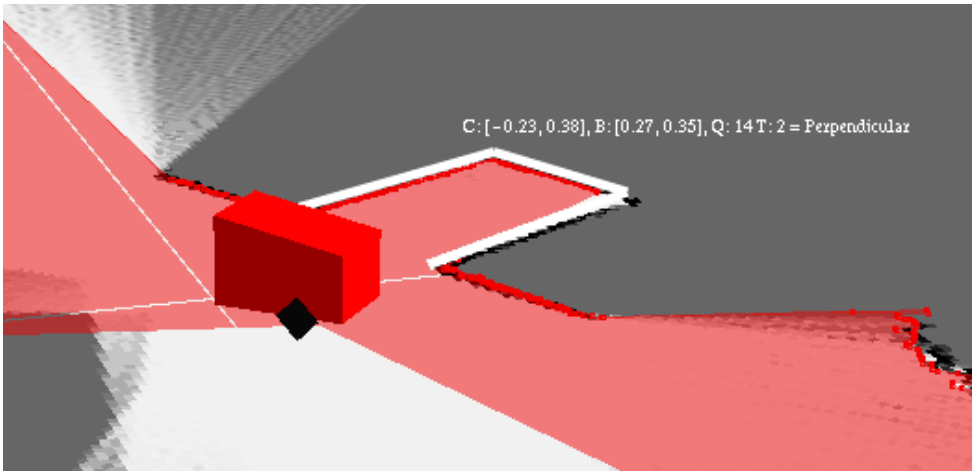
Takto nalezená parkovací místa jsou nyní vložena do kvadrantového stromu. Před tím, než se parkovací místo vloží do stromu, se zkontroluje, zda poblíž středu vkládáného parkovacího



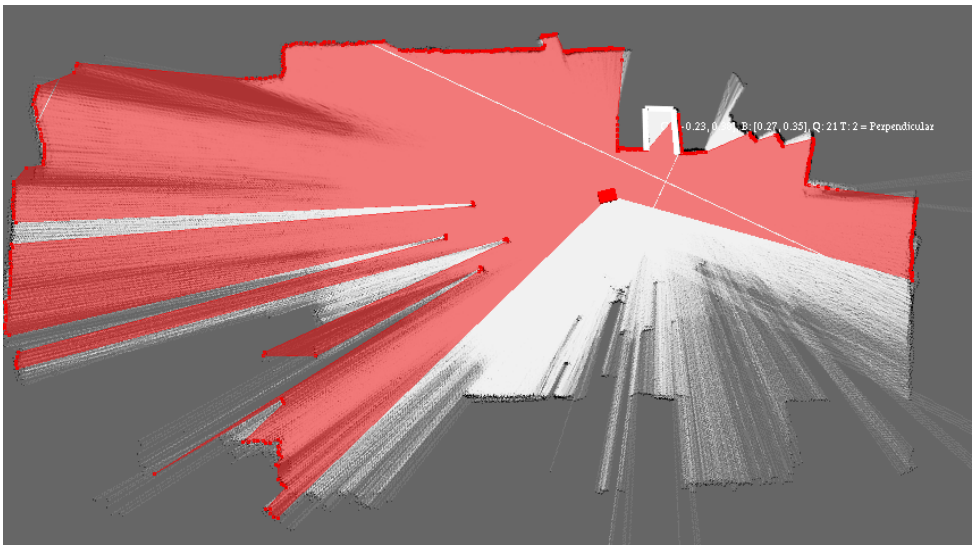
Obrázek 25: Diagram aktivit algoritmu pro detekci parkovacího místa

místa není již uloženo jiné místo. Pokud ano, a zároveň odchylky vzdáleností korespondujících rohových bodů jsou nízké, je toto již uložené místo aktualizováno a inkrementována hodnota určující kvalitu místa, resp. vyjadřující počet výskytů. Pokud v okolí středu vkládaného místa nebylo nalezeno žádné jiné parkovací místo, je vkládané místo přímo vloženo do stromu. Tímto způsobem je zajištěna aktualizace již rozpoznaných míst. Místa s nízkou kvalitou, resp. nízkým počtem výskytů, jsou označena jako špatně vyhodnocená a nejsou použita pro parkování.

Výsledek rozpoznávání je vizualizován na obrázku 26, kde je zobrazen model automobilu v trojrozměrném prostoru společně s mřížkou obsazenosti (podlahová rovina), aktuálním skenem (červený polygon ohraničen body) a rozpoznaným parkovacím místem, které je ohraničeno bílými úsečkami a označeno textem s doplňujícími informacemi o středu C, velikosti B, kvalitě Q a typu místa T (jednotky v metrech). Na obrázku 27 je další úkazka nalezeného parkovacího místa.



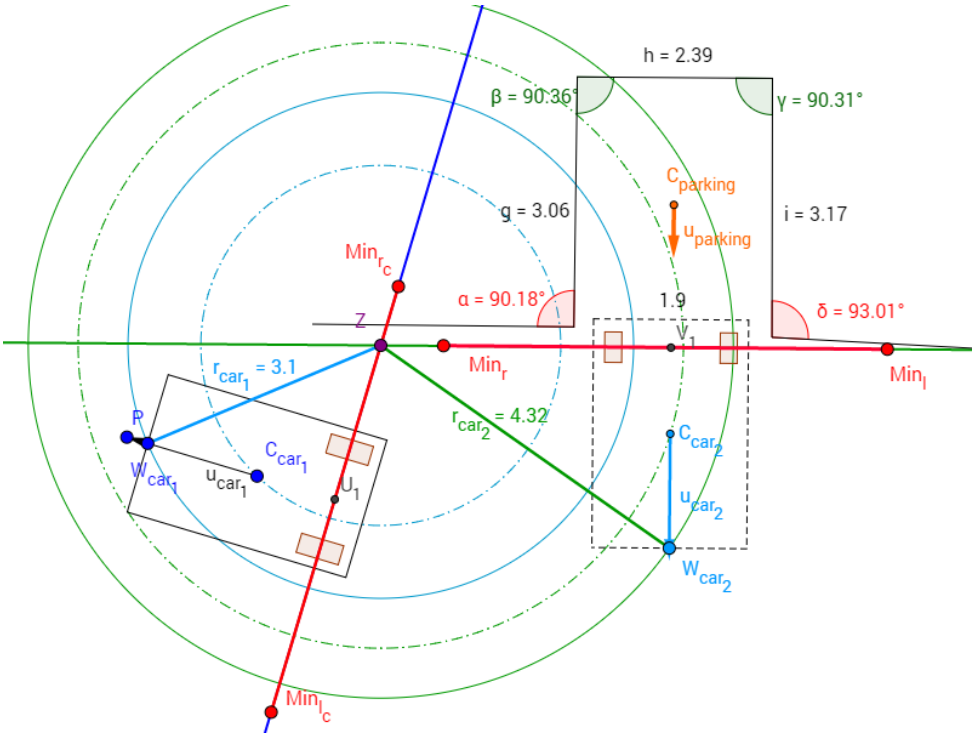
Obrázek 26: Ukázka vizualizace okolí automobilu společně s vyhodnoceným parkovacím místem



Obrázek 27: Vizualizace mapy okolí s vyhodnoceným parkovacím místem

8 Matematický model pro automatické parkování

V předchozích kapitolách bylo popsáno, jakým způsobem hlavní počítač ovládá model automobilu, jak pracuje Ackermannův podvozek, jak hlavní počítač získává snímky okolí ze skeneru, jak je určena poloha v mapě okolí, která se aktualizuje s každým dalším snímkem, a jak se v této mapě rozpozná parkovací místo. Hlavní počítač má nyní veškeré potřebné informace a nástroje k tomu, aby byl schopen řídit model automobilu tak, aby zaparkoval na rozpoznané parkovací místo.



Obrázek 28: Geometrie parkování pro kolmé parkovací místo

8.1 Kolmé parkování

Pro zaparkování automobilu na kolmé parkovací místo je zapotřebí automobilem zajet před parkovací místo, na obrázku 28 je to čárkovaně vyznačný obdélník, a následně zajet dovnitř parkovacího místa.

Na obrázku 28 jsou vyznačeny geometrické útvary určující způsob pohybu automobilu při parkování. Automobil je označen černým obdélníkem se středem C_{car_1}. Směr automobilu je dán vektorem u_{car_1}. Dále je na obrázku osa zadní nápravy automobilu (modrá přímka) a na této ose jsou vyznačeny červenou úsečkou poloměry otáčení, kterých nelze docílit (červená úsečka na ose zadní nápravy omezená body Min_{l_c} a Min_{r_c}). Dále je na obrázku vyznačena požadovaná poloha automobilu jako čárkovaný obdélník se středem C_{car_2} a směrem u_{car_2}. Také je zde zobrazena osa

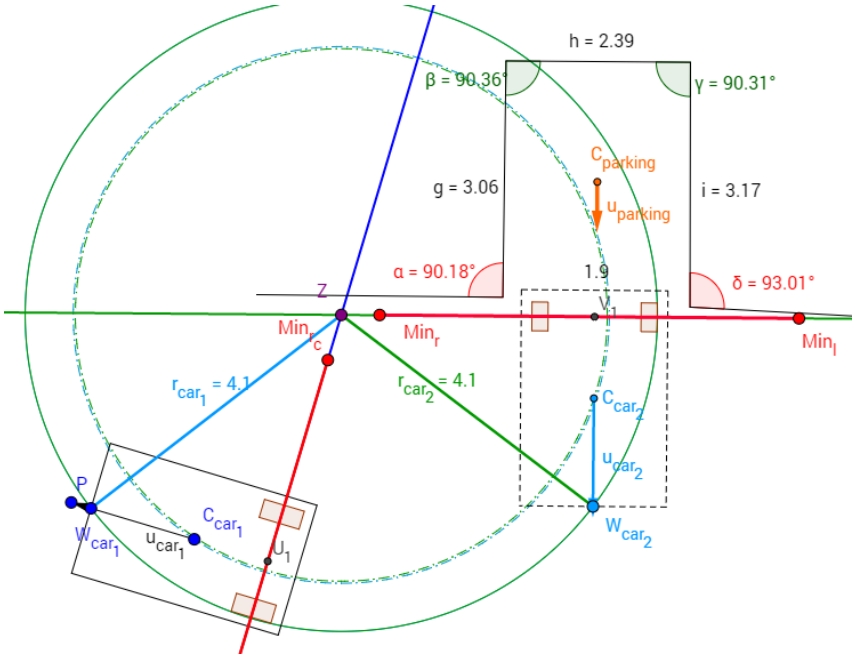
zadní nápravy jako zelená přímka a na ní jsou vyznačeny nedovolené poloměry otáčení červenou úsečkou omezenou body Min_r a Min_l. Tyto body jsou určeny minimálními poloměry otáčení viz tabulka 3 a vztahy 20, 21:

$$|Min_rS| = \sqrt{r_{min_r}^2 - l^2}, \tag{20}$$

$$|Min_lS| = \sqrt{r_{min_l}^2 - l^2}, \tag{21}$$

kde S je střed osy zadní nápravy, l je rozvor náprav, r_{min_l} je minimální poloměr otáčení vlevo a r_{min_r} vpravo. Tyto poloměry byly naměřeny a jejich hodnoty odpovídají r_i v tabulce 3 u hodnot W -1000 a 1000.

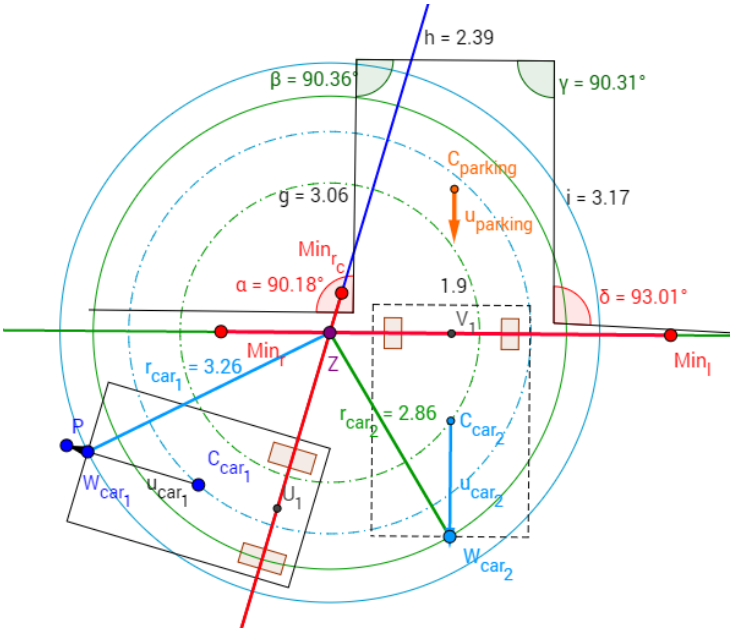
Průsečík osy zadní nápravy automobilu s osou zadní nápravy požadované polohy je vyznačen bodem Z. Z průsečíku Z jsou vyneseny kružnice procházející středem automobilu (čerchovaná čára) a středem pomyslného prostředního kola (plná čára). Poloměry těchto kružnic (r_{car_1} a r_{car_2}) jsou v úloze parkování klíčové. Tyto kružnice vyznačují dráhu pohybu automobilu. Pokud jsou si poloměry těchto kružnic rovny (resp. přibližně rovny), pohybem automobilu směrem dozadu s nastaveným poloměrem otáčení na hodnotu r_{car_1} nebo r_{car_2} se automobil dostane na požadovanou polohu (na místo čárkovaného obdélníku). Ukázka takového případu je na obrázku 29.



Obrázek 29: Geometrie parkování při nalezeném poloměru otáčení

Průsečík Z nesmí ležet na červených úsečkách viz obrázek 30, které určují nedovolené poloměry otáčení. Pokud by průsečík Z byl na těchto místech, je potřeba zařídit pohybem automobilu, aby tomu tak nebylo. Leží-li průsečík Z na úsečce $Min_{r_c}Min_{l_c}$ je nutné jet s automobilem vertikálním směrem od parkovacího místa. Je-li průsečík Z na úsečce Min_rMin_l je nutné jet

s automobilem horizontálním směrem od parkovacího místa. V případě, kdy leží průsečík na obou úsečkách, je nutno jet s automobilem ve směru od parkovacího místa. Tímto způsobem se zajistí opuštění zón nedovolených poloměrů otáční. V případě obrázku 30 se bude automobil vzdalovat od parkovacího místa.

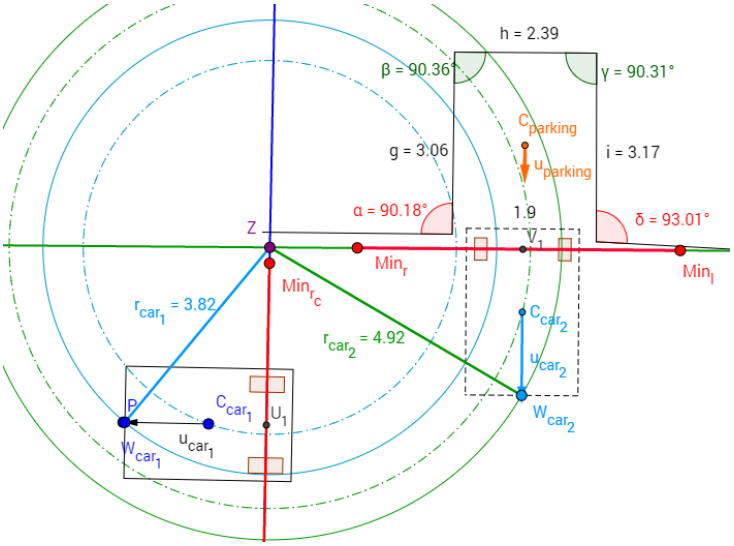


Obrázek 30: Geometrie parkování při nutnosti posunu automobilu k opuštění zón s nedovoleným poloměrem otáčení

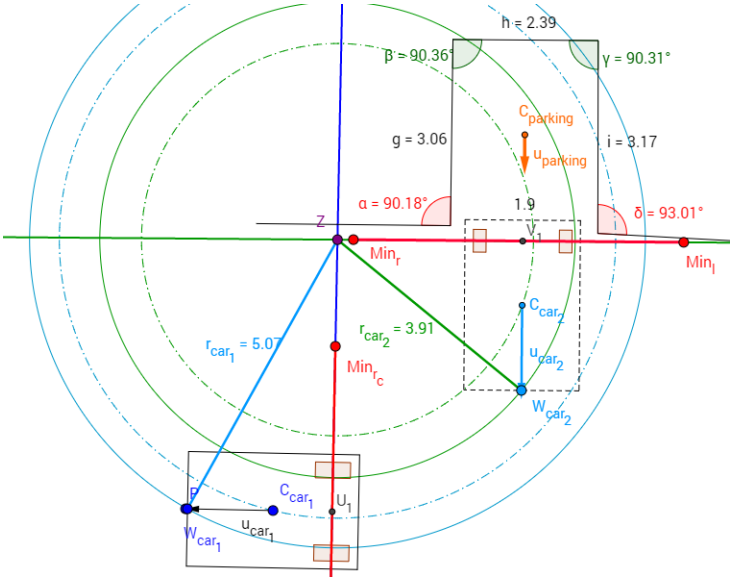
V momentě, kdy automobil opustil zóny s nedovolenými poloměry zatáčení, lze začít s posunem automobilu tak, aby byl nalezen poloměr otáčení pro správné zaparkování. Nejsou-li si poloměry r_{car_1} a r_{car_2} rovny, je potřeba pohybovat s automobilem tak, aby se hodnoty těchto poloměrů přibližovaly. Na obrázku 31 je zobrazen tento případ. Je-li poloměr $r_{car_1} < r_{car_2}$, je zapotřebí posunout automobil v horizontálním směru k parkovacímu místu, nebo ve vertikálním směru od parkovacího místa. V opačném případě $r_{car_1} > r_{car_2}$ viz obrázek 32, je zapotřebí posunout automobil v horizontálním směru od parkovacího místa, nebo ve vertikálním směru k parkovacímu místu [12, 13].

8.1.1 Algoritmus parkování na kolmé parkovací místo

Výše vysvětlené situace tvoří části algoritmu parkování. Algoritmus je v počátečním stavu 1, ve kterém je kontrolováno, zda-li neleží průsečík Z na úsečkách určující nedovolené poloměry zatáčení viz obrázek 30. Pokud průsečík Z leží na těchto úsečkách je zahájen pohyb automobilu. V případě, kdy průsečík leží na úsečce $Min_{r_c}Min_{l_c}$, je automobil veden vertikálním směrem od parkovacího místa. Pokud průsečík leží na úsečce Min_rMin_l, je automobil veden horizontálním směrem od parkovacího místa. Leží-li průsečík na obou úsečkách, je automobil veden směrem od



Obrázek 31: Geometrie parkování při nutnosti posunu ($r_{car_1} < r_{car_2}$)



Obrázek 32: Geometrie parkování při nutnosti posunu ($r_{car_1} > r_{car_2}$)

parkovacího místa. Pokud průsečík Z opustil úsečky nedovolených poloměrů zatáčení, přechází algoritmus do stavu 2.

Algoritmus se ve stavu 2 snaží docílit takové polohy automobilu, aby se poloměry otáčení r_{car_1} a r_{car_2} přibližně rovnaly. V případě, který je zobrazen na obrázku 31, kdy je $r_{car_1} < r_{car_2}$, jsou kola automobilu vytočena tak, aby se směr automobilu vyrovnal se směrem vektoru u_{car_2} a automobil je veden dopředu. Tímto se zajistí posunutí automobilu od osy zadní nápravy požadované polohy automobilu, a tudíž i dojde ke zvýšení poloměru r_{car_1}. V případě, který je zobrazen na obrázku 32, kdy je $r_{car_1} > r_{car_2}$ jsou kola automobilu natočena ve směru vektoru u_{car_2} natočeného o 90° doprava a automobil je veden dopředu. Tímto se zajistí zvýšení poloměru

r_{car_2}. Tímto vedením automobilu je zajištěno vyrovnávání poloměrů r_{car_1} a r_{car_2}.

Vztah pro vyrovnání automobilu do směru, který je určen vektorem u je dán funkcí 22:

$$f(u,v) = \left[v_x \left(u_x \cos\left(\frac{\pi}{2}\right) - u_y \sin\left(\frac{\pi}{2}\right) \right) + v_y \left(u_x \sin\left(\frac{\pi}{2}\right) + u_y \cos\left(\frac{\pi}{2}\right) \right) \right] M_\alpha, \qquad (22)$$

kde vektor v je aktuální směrový vektor automobilu a konstanta M_α maximální úhel zatáčení pomyslného prostředního kola. Výstupem této funkce je úhel natočení tohoto kola. Tohoto vztahu je využíváno právě pro pohyb automobilu např. ve směru u_{car_2} v horizontálním, resp. vertikálním směru od parkovacího místa.

V momentě, kdy jsou poloměry $r_{car_1} \approx r_{car_2}$, je automobil veden dozadu s nastaveným poloměrem otáčení r, který je roven aritmetickému průměru poloměrů r_{car_1} a r_{car_2}.

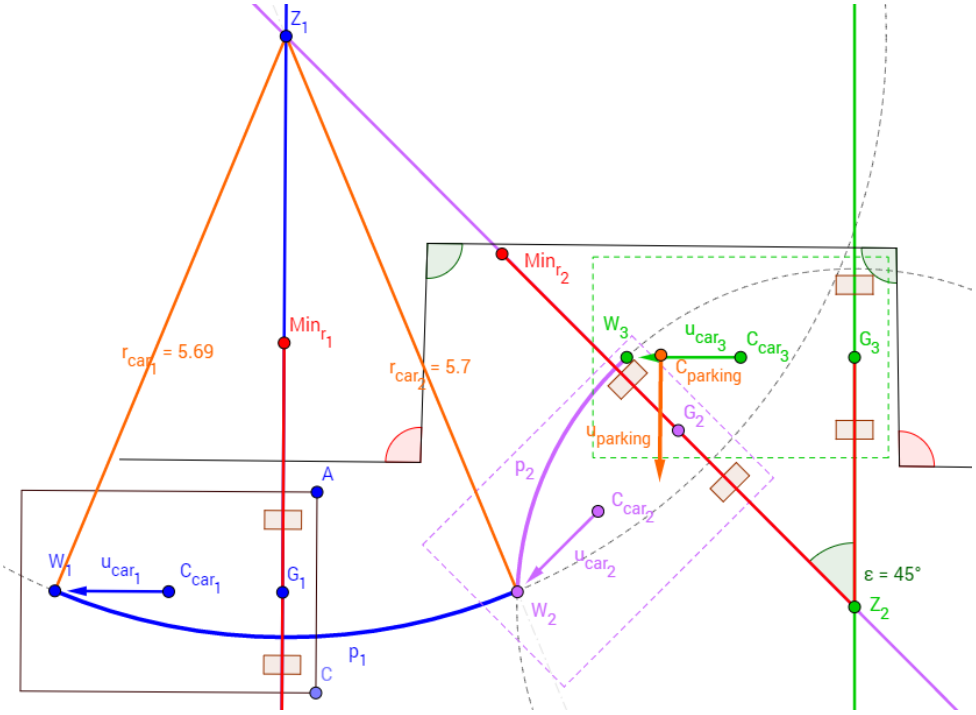
Tyto tři situace zajišťují přesun automobilu na požadovanou polohu před parkovací místo. Když se této polohy automobilu docílí, resp. jsou-li body C_{car_1} a C_{car_2} dostatečně blízko, přechází algoritmus do stavu 3.

Algoritmus ve stavu 3 zajišťuje vedení automobilu dozadu, a to tak, aby se směr automobilu u_{car_1} vyrovnával se směrem $u_{parking}$ pomocí funkce 22. V momentě, kdy jsou body C_{car_1} a $C_{parking}$ dostatečně blízko sebe je automobil zastaven a parkovací proces dokončen. Průběh parkování je zaznamenán na obrázcích 36 v příloze A.

8.2 Podélné parkování

U podélného parkování je zapotřebí automobilem zajet na místo vyznačené fialovým čárkovaným obdélníkem viz obrázek 33, z tohoto místa zajet na místo vyznačené zeleným čárkovaným obdélníkem a nakonec popojet do středu parkovacího místa $C_{parking}$. Pozice automobilu je na obrázku 33 označena černým obdélníkem s vyznačeným středem C_{car_1} a vektorem určující natočení automobilu u_{car_1}. Také je zobrazena osa zadní nápravy jako modrá přímka se středem G_1 a středem pomyslného prostředního kola W_1. Takto jsou označeny také všechny požadované pozice automobilu. První požadovaná pozice je označena fialovým čárkovaným obdélníkem se středem C_{car_2}. Natočení první požadované pozice je určeno vektorem u_{car_2}. Dále je také zobrazena osa zadní nápravy jako fialová přímka a její střed G_2. Střed pomyslného prostředního kola je v tomto případě W_2. Stejné značení je použito u druhé požadované polohy automobilu, u které je zvolena zelená barva a u značení jsou indexy s hodnotou 3. První požadovaná pozice automobilu, která je zobrazena jako fialový čárkovaný obdélník, je posunutá druhá požadovaná pozice, která je zobrazena jako zelený čárkovaný obdélník, o úhel $\varepsilon = 45°$ pohybem dopředu s nastaveným poloměrem otáčení na levý minimální r_{min_l}, jehož hodnota odpovídá r_i z tabulky 3 u hodnoty W -1000.

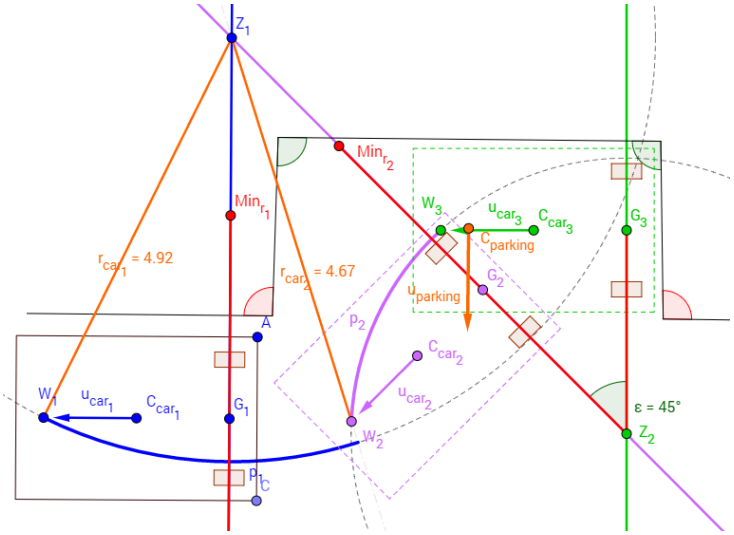
Na osách zadních náprav jsou označeny červenou úsečkou nedovolené středy otáčení, stejně jako u kolmého parkování. Body Min_{r_i} a Min_{l_i} jsou určeny vztahy 20 a 21, kde S je vždy střed osy zadní nápravy. Stejně jako u kolmého parkování jsou zde vyznačeny průsečíky os



Obrázek 33: Geometrie parkování pro podélné parkovací místo

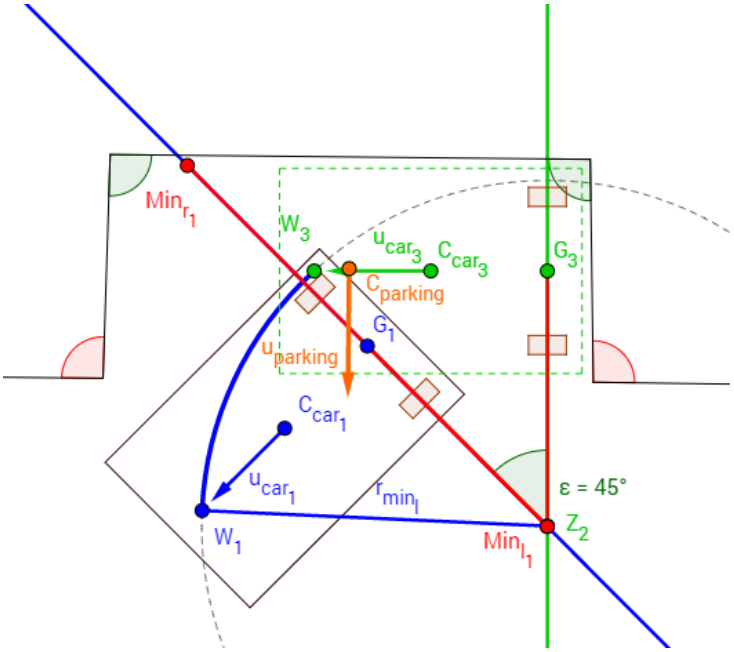
zadních náprav Z_1, Z_2. Vzdálenosti mezi středy pomyslného předního kola a průsečíkem Z_1 jsou označeny $r_{car_1} = |W_1Z_1|$ a $r_{car_2} = |W_2Z_1|$. Tyto vzdálenosti hrají opět klíčovou roli v řešení úlohy parkování. K nalezení takového poloměru otáčení, který by automobil posunul na první požadovanou polohu, je nutno pohybovat s automobilem tak, aby si poloměry r_{car_1} a r_{car_2} byly přibližně rovny. Ukázka nalezeného poloměru otáčení, který posune automobil na první požadovanou pozici je na obrázku 33.

K nalezení poloměru otáčení, který by posunul automobil do první požadované polohy, je potřeba, aby se automobil dostal co nejblíže k parkovacímu místu, a současně se pohyboval dopředu, resp. dozadu ve směru vektoru u_{car_3}. Na obrázku 34 je automobil blízko parkovacímu místu a nasměrován podle vektoru u_{car_3}. Pokud se nyní bude automobil pohybovat směrem dopředu, budou se hodnoty poloměrů r_{car_1} a r_{car_2} zvyšovat společně, ale r_{car_2} bude stoupat rychleji vůči r_{car_1}. Při pohybu dozadu se budou hodnoty poloměrů r_{car_1} a r_{car_2} společně zmenšovat, ale r_{car_2} bude klesat rychleji vůči r_{car_1}. Lze tedy tvrdit, že pokud je $r_{car_1} < r_{car_2}$, je potřeba k nalezení poloměru otáčení, který by automobil dostal na první požadovanou pozici, pohybovat s automobilem dozadu, a pokud je $r_{car_1} > r_{car_2}$, je potřeba pohybovat s automobilem dopředu [13].



Obrázek 34: Geometrie podélného parkování, kde je automobil nejblíže k parkovacímu místu

Ve chvíli, kdy je automobil na první požadované pozici viz obrázek 35, je pohybováno automobilem dozadu s nastaveným poloměrem otáčení na r_{min_l}. Nakonec je s automobilem pohybováno dopředu ve směru vektoru $u_{parking}$ natočeného o 90° doprava. Automobil je zastaven, jakmile je střed automobilu na pozici středu parkovacího místa $C_{parking}$ (resp. přibližně na $C_{parking}$) [12].



Obrázek 35: Geometrie podélného parkování, kde je automobil na první požadované pozici

8.2.1 Algoritmus parkování na podélné parkovací místo

V první fázi algoritmu je automobil veden co nejblíže k parkovacímu místu, a také je směr automobilu vyrovnáván ve směru vektoru u_{car_3}. Ve chvíli, kdy je automobil dostatečně blízko a platí $u_{car_1} \approx u_{car_3}$, algoritmus přechází do stavu 2.

V druhém stavu algoritmus pohybuje autem ve směru u_{car_3} dopředu, pokud platí $r_{car_1} > r_{car_2}$ resp. dozadu, pokud platí $r_{car_1} < r_{car_2}$. Pokud jsou poloměry přibližně stejné $r_{car_1} \approx r_{car_2}$ je automobil veden dozadu s poloměrem otáčení, jehož hodnota je aritmetickým průměrem r_{car_1} a r_{car_2}. V momentě, kdy se střed automobilu C_{car_1} dostane na pozici středu první požadované pozice C_{car_2}, algoritmus přechází do stavu 3.

Ve třetím stavu algoritmu se nachází automobil v první požadované pozici viz obrázek 35. Nyní je automobil veden dozadu s nastaveným poloměrem otáčení na r_{min_l}. Ve chvíli, kdy bude střed automobilu C_{car_1} na pozici středu druhé požadované pozice C_{car_3}, algoritmus přechází do stavu 4.

Ve čtvrtém stavu algoritmu je automobil veden dopředu ve směru u_{car_3}, aby došlo k dorovnání automobilu a posunutí na střed parkovacího místa $C_{parking}$. Jakmile se střed automobilu C_{car_1} dostane na pozici středu parkovacího místa $C_{parking}$ je proces parkování dokončen a algoritmus ukončen. Průběh parkování je zaznamenán na obrázcích 37 v příloze A.

9 Vyhodnocení parkovacích modelů

Pro vyhodnocení úspěšnosti parkovacích modelů bylo uměle vytvořeno parkovací místo z lepenkových krabic o předem určených rozměrech. Automobilem bylo v rámci jednoho měření pohybováno po stejných trasách pro naskenování okolí a nalezení parkovacího místa. Následně byla testována schopnost zaparkování. Toto měření bylo vždy desetkrát zopakováno. Výstupem jednoho měření bylo, zda automobil při parkování narazil do překážky, a zda automobil úspěšně zaparkoval. Za neúspěch zaparkování je považováno zaseknutí mezi stavy algoritmu na více než 10 vteřin a v případě, kdy dané parkovací místo nebylo rozpoznáno je vyhodnocen neúspěch.

Rozměry modelu automobilu jsou 19×28 cm. K těmto rozměrům byla přidána tolerance 1 cm na všechny strany, takže parkovací místa menší než 21x30 cm, by neměla být vyhodnocena jako vhodná pro zaparkování.

9.1 Výsledky kolmého parkování

Pro testování kolmého parkování byly použity rozměry parkovacího místa 21×30 cm, 22×30 cm, 25x32 cm a 30×35 cm. Výsledky jsou uvedeny v tabulce 4.

Tabulka 4: Výsledky testování modelu pro kolmé parkování

Rozměry místa (cm)	**Rozpoznaná místa**	**Úspěchy**	**Kolize**[1]	**Úspěšnost (%)**
30×35	10	10	0	100
25×32	10	10	1	100
22×30	10	8	2	80
21×30	7	7	4	70

Při menších rozměrech parkovacího místa měl algoritmus pro parkování problémy a to hlavně z důvodů vysokého zkreslení skeneru, který nastává při skenování blízkých vzdáleností a k tomu dochází při vjíždění automobilu do parkovacího místa. Při vysoce zkresleném snímku okolí dochází ke špatnému určení polohy automobilu algoritmem ICP-SLAM. Také se stávalo v některých pokusech, že algoritmus nedokázal najít poloměr otáčení, a tím docházelo k oscilování automobilu kolem určité polohy. Příčinou tohoto zaseknutí mohlo být malé rozlišení mřížky obsazenosti.

[1]Sloupec kolize uvádí v kolika pokusech automobil narazil do překážky.

9.2 Výsledky podélného parkování

Pro testování podélného parkování byly použity rozměry parkovacího místa 30×21 cm, 32×22 cm, 34×25 cm, 38×27 cm a 40×30 cm. Výsledky jsou uvedeny v tabulce 5.

Tabulka 5: Výsledky testování modelu pro podélné parkování

Rozměry místa (cm)	**Rozpoznaná místa**	**Úspěchy**	**Kolize**[1]	**Úspěšnost (%)**
40×30	10	10	2	100
38×27	10	9	2	90
34×25	10	6	5	40
32×22	10	2	10	20
30×21	8	0	10	0

U rozměru 30×21 cm nebylo možno zaparkovat vzhledem k minimálnímu poloměru otáčení. U menších parkovacích míst docházelo k častým kolizím s překážkami. Také zde docházelo u některých pokusů k zasekávání algoritmu mezi stavy jako u kolmého parkování.

[2]Sloupec kolize uvádí v kolika pokusech automobil narazil do překážky.

10 Závěr

Cílem mé diplomové práce bylo vytvořit řízení modelu automobilu s Ackermannovým podvozkem, které řídí automobil tak, aby automaticky zaparkoval na kolmé či podélné parkovací místo. Cíl práce byl splněn a úspěšnosti parkovacích modelů odpovídají očekáváním.

Na začátku vytváření této práce jsem vymýšlel způsob uchycení komponent na model automobilu. Poté jsem se seznamoval s řízením použitého modelu automobilu. Následně jsem se seznámil s komunikačním protokolem skeneru Hokuyo URG-04LX a vyzkoušel detekci objektů v okolí. Pak jsem se seznámil s technikami SLAM, konkrétně s algoritmem ICP-SLAM, který jsem použil pro vytváření mapy okolí a lokalizaci automobilu. Následovalo použití mřížky obsazenosti pro reprezentaci mapy okolí a snížení paměťové složitosti. Nakonec jsem vytvářel matematické modely pro oba typy parkování. Kreslil jsem mnoho náčrtů geometrie parkování a dával do souvislosti informace, které mi pomohly s vytvořením matematického modelu pro parkování. Navržené algoritmy vyplývající z těchto prvotních modelů byly v prvních fázích velmi nedokonalé a jejich úspěšnost byla sotva 10%. Druhá verze vytvořených algoritmů pro parkování měla založenou myšlenku na jiných matematických modelech a jejich účinnost byla ještě menší. Ve třetí verzi algoritmů, bylo využíváno jednoduché heuristiky. Tato verze ovšem vedla k výsledkům, které nebyly o nic lepší.

Po pěti vytvořených verzích algoritmů (resp. matematických modelů) začaly algoritmy (resp. matematické modely) mít relativně vysokou úspěšnost. Konečné dolazení parametrů ICP-SLAM algoritmu a parametrů ovlivňující vytváření úseček pro detekci parkovacího místa zvýšily úspěšnost.

Průběh práce stěžoval vysoký odběr komponent a motorů automobilu, díky němuž byla 3000mAh baterie vybita během dvou hodin. Kompilace knihoven a utilit pro Raspberry Pi trvala celkem přibližně 40 hodin.

Literatura

[1] Uživatelská příručka k FRDM-K64F. *FRDM-K64F Freedom Module User's Guide* [online]. Evropa: NXP Semiconductors N.V. 2016 [cit. 2017-04-15]. Dostupné z: http://www.nxp.com/assets/documents/data/en/user-guides/FRDMK64FUG.pdf

[2] Specifikace skeneru Hokuyo URG-04LX. *Scanning Laser Range Finder URG-04LX* [online]. Japonsko: HOKUYO AUTOMATIC CO., LTD 2005 [cit. 2017-04-15]. Dostupné z: http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_spec_en.pdf

[3] Freescale Cup Shield for the Freedom KL25Z. *Freescale Cup Shield for the Freedom KL25Z | NXP Community* [online]. [cit. 2017-04-15]. Dostupné z: https://community.nxp.com/docs/DOC-93914

[4] Ackermann steering geometry. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-04-15]. Dostupné z: https://en.wikipedia.org/wiki/Ackermann_steering_geometry

[5] Lidar. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2017-04-15]. Dostupné z: https://en.wikipedia.org/wiki/Lidar

[6] Laser rangefinder. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2017-04-15]. Dostupné z: https://en.wikipedia.org/wiki/Laser_rangefinder

[7] Simultaneous localization and mapping. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2017-04-15]. Dostupné z: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

[8] Specifikace komunikačního protokolu pro SCIP2.0 standard. In: *Communication Protocol Specification For SCIP2.0 Standard* [online]. 2006 [cit. 2017-04-15]. Dostupné z: http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG_SCIP20.pdf

[9] Weisstein, Eric W. "Least Squares Fitting–Perpendicular Offsets."In: *MathWorld–A Wolfram Web Resource* [online]. [cit. 2017-04-15]. Dostupné z: http://mathworld.wolfram.com/LeastSquaresFittingPerpendicularOffsets.html

[10] Design of an Ackermann-type steering mechanism [online]. [cit. 2017-04-15]. Dostupné z: https://www.researchgate.net/publication/265755401_Design_of_an_Ackermann_Type_Steering_Mechanism

[11] Automatic Steering Methods for Autonomous Automobile Path Tracking [online]. [cit. 2017-04-15]. Dostupné z: https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf

[12] Autonomous Parallel Parking Methodology for Ackerman Configured Vehicles [online]. [cit. 2017-04-15]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.394.2446&rep=rep1&type=pdf

[13] Development of automated parallel parking system in small mobile vehicle [online]. [cit. 2017-04-15]. Dostupné z: http://www.arpnjournals.com/jeas/research_papers/rp_2015/jeas_0915_2526.pdf

[14] Autonomous Parking System Techniques [online]. [cit. 2017-04-15]. Dostupné z: http://ijiere.com/FinalPaper/FinalPaperAutonomous%20Parking%20System%20Techniques%20%E2%80%93%20A%20Review170529.pdf

A Obrázky a fotografie

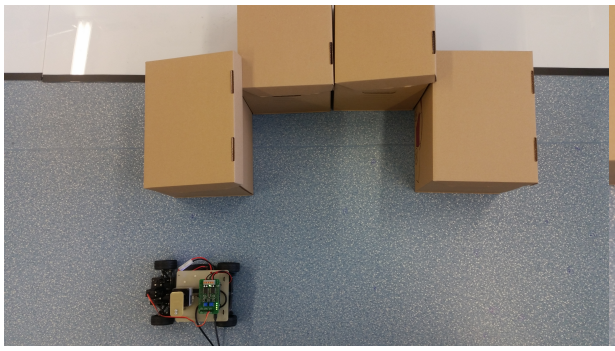
(a) Začátek parkování

(b) Pohyb na požadovanou pozici před parkovacím místem

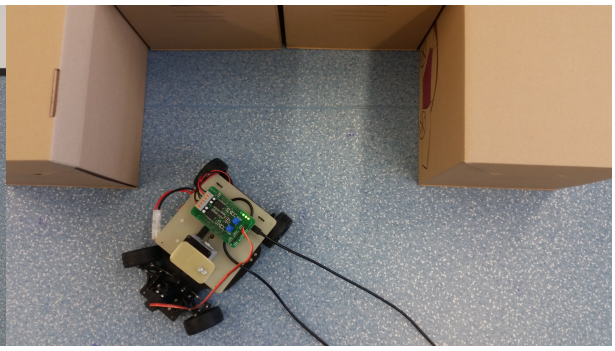
(c) Na požadované pozici před parkovacím místem

(d) Po zaparkování

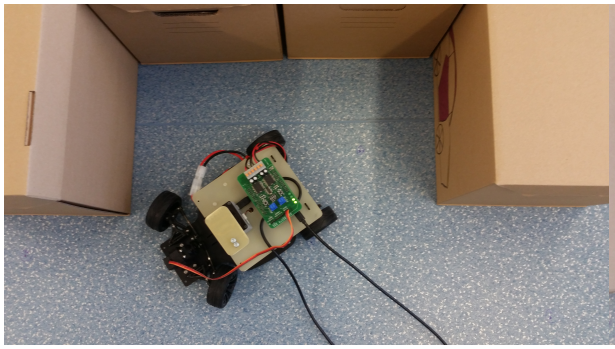
Obrázek 36: Průběh parkování na kolmé parkovací místo



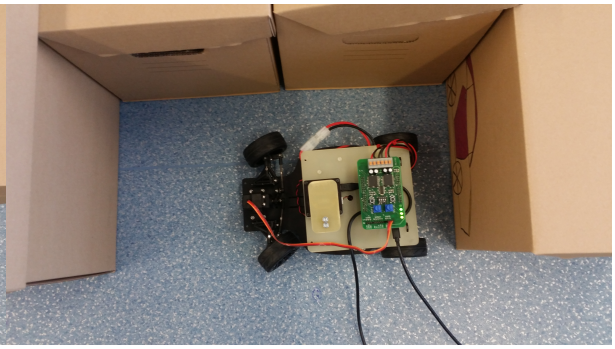
(a) Začátek parkování



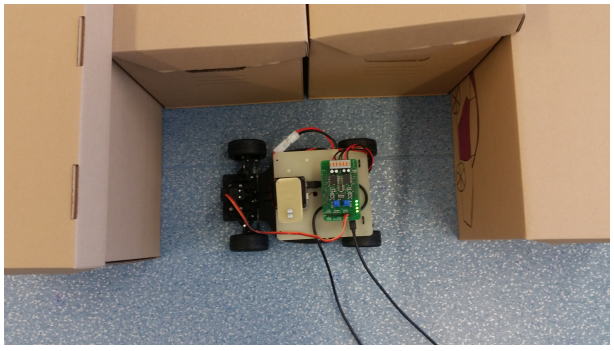
(b) Pohyb na první požadovanou pozici



(c) Pohyb na druhou požadovanou pozici



(d) Na druhé požadované pozici



(e) Po zaparkování

Obrázek 37: Průběh parkování na podélné parkovací místo

B Výpisy zdrojových kódů

V této sekci jsou výpisy delších zdrojových kódů.

```
// commands
#define CMD_DATA      1
#define CMD_SETTING   2
#define CMD_CONTROL   3

// protocol start and stop byte
#define STX           0x2
#define ETX           0x3

struct s_protocol_empty
{
   uint8_t    stx;
   uint16_t   length;
   uint8_t    cmd;
   uint8_t    etx;
} __attribute__((packed));

struct s_data
{
   uint32_t   timestamp;            // number of sample
   uint16_t   adc[ anLast ];        // ADC channels
   uint8_t    dip_sw;               // dip switches status
   uint8_t    push_sw;           // push buttons
   uint16_t   image[ CAMERA_LINE_LENGTH ];// line camera image
   uint32_t   _padding;
} __attribute__((packed));

struct s_setting
{
   uint16_t   servo_center[ 2 ];      // servos calibration
   uint16_t   servo_max_lr[ 2 ];      // servos range
   uint16_t   pwm_max;           // maximal PWM for motors
   uint16_t   _padding;
} __attribute__((packed));

struct s_control
```

```
{
   uint8_t    leds;              // four leds - low nibble
   uint8_t    pwm_onoff;            // enable motors <0,1>
   uint8_t    servo_onoff;          // enable servo <0,1>
   uint8_t    _padding1;
   int16_t    pwm_a, pwm_b;         // PWM of motors <-PWM_MINMAX, PWM_MINMAX>
   int16_t    servo_pos[ 2 ];          // servos position <-SERVO_MAX_LR,
      SERVO_MAX_LR>
} __attribute__((packed));
```

Výpis 1: Definice struktur pro komunikaci s FRDM-K64F

```
   void FindLines(vector<LineSegment> & lines, const double angleOffset) const
      {

      vector<Point2> linePoints;
      bool endOfLineDetected = false;
      Point2 prevPoint;
      double prevDistanceBetweenPrevPoint = 0;
      LineSegment prevLine;

      // in case lidarScan->rightToLeft == true
      const int lidarScanSize = lidarScan->scan.size();

      for (int i = lidarScanSize - 1; i >= 0; i--) {

         if (!lidarScan->validRange[i]) {
            if (linePoints.size() > 1) {
               LineSegment line = LeastSquaresLine(linePoints);

               line.start = line.PerpendicularPoint(linePoints[0]);
               line.end = line.PerpendicularPoint(linePoints.back());

               lines.push_back(line);
               linePoints.clear();
            }
            continue;
         }
```

```
double angle = hokuyoURG04LXAngleDiff * ((lidarScanSize - 1) - i) +
    hokuyoURG04LXStartAngle + angleOffset;
double r = lidarScan->scan[i] * meterToPixelRatio;

double x = (cos(angle * M_PI / 180.0) * r);
double y = (sin(angle * M_PI / 180.0) * r);

Point2 p = Point2(x + lidarScan->maxRange * meterToPixelRatio, y +
    lidarScan->maxRange * meterToPixelRatio);

if (linePoints.size() == 0)
   linePoints.push_back(p);
else {
   double distanceToPrevPoint = prevPoint.Distance(p);
   if (distanceToPrevPoint > distanceThresh) {
      // end of line
      if (linePoints.size() > 1) {
         LineSegment line = LeastSquaresLine(linePoints);

         line.start = line.PerpendicularPoint(linePoints[0]);
         line.end = line.PerpendicularPoint(linePoints.back());

         lines.push_back(line);
      }
      linePoints.clear();
      linePoints.push_back(p);
      prevDistanceBetweenPrevPoint = p.Distance(prevPoint);
      prevPoint = p;
      continue;
   }
   else {
      linePoints.push_back(p);
   }
}

if (linePoints.size() > 1) {
   LineSegment line = LeastSquaresLine(linePoints);
```

```
      if (abs(line.r - prevLine.r) > errorThresh && linePoints.size() >
          linePointsThresh) {

         line.start = line.PerpendicularPoint(linePoints[0]);
         line.end = line.PerpendicularPoint(linePoints.back());

         if (lines.size() > 1) {
            LineSegment & backLine = lines[lines.size() - 1];

            if (backLine.end.Distance(line.start) < intersectThresh) {

               Point2 intersect = backLine.Intersection(line);

               if (intersect.Distance(backLine.end) < intersectThresh) {
                  backLine.end = intersect;
                  line.start = intersect;
               }
            }
         }

         lines.push_back(line);

         linePoints.clear();
         linePoints.push_back(prevPoint);

         prevLine = line;
      }
      else {
         prevLine = line;
      }
   }
   prevDistanceBetweenPrevPoint = p.Distance(prevPoint);
   prevPoint = p;
}

if (linePoints.size() > 1) {
   LineSegment line = LeastSquaresLine(linePoints);

   line.start = line.PerpendicularPoint(linePoints[0]);
```

```
      line.end = line.PerpendicularPoint(linePoints.back());

      lines.push_back(line);

      linePoints.clear();
    }

    for (int i = 1; i < lines.size(); i++) {
      LineSegment & line = lines[i];
      LineSegment & prev = lines[i - 1];

      Point2 intersect = prev.Intersection(line);

      double thresh = smallLineThresh * 1.5;

      if (line.start.Distance(prev.end) < thresh) {
        if (intersect.Distance(line.start) < thresh * 2) {
          prev.end = intersect;
          line.start = intersect;
        }
        else {
          // nothing to do
        }
      }
    }
  }
```

Výpis 2: Definice funkce pro vytvoření úseček ze vzdáleností naměřených skenerem

```
    struct CornerHelper {
      Point2 corner;
      bool negative;
    };
    vector<CornerHelper> foundCorners;

    vector< vector<Point2> > corners;
    bool prevAngleNegative = false; // true means left

    for (int i = 0; i < lines.size() - 1; i++) {
      LineSegment & line = lines[i];
```

```
if (line.Length() < smallLineThresh) {
   continue;
}

int j = 1;
for (; j <= maximumNextLines && j + i < lines.size(); j++) {
   LineSegment & nextLine = lines[i + j];

   if (nextLine.Length() < smallLineThresh) {
      // small lines are skipped
      continue;
   }

   Point2 intersection = line.Intersection(nextLine);

   // if distance between intersection of line and nextLine to line
       endpoint - in other words: intersection is not far away
   if (intersection.Distance(line.end) < cornerIntersectionMaxDistance)
       {
      double cornerAngle = line.AngleBetween(nextLine);
      if (abs(cornerAngle) >= cornerAngleThresh) {
         foundCorners.push_back({ intersection, cornerAngle < 0 });
         break;
      }
   }
}

if (foundCorners.size() == 4) {
   i += j - 1;

   // if angles of corners are + - - + or - + + - = parking place
   CornerHelper * h = &foundCorners[0];
   if (h->negative) {
      h = &foundCorners[1];
      if (!h->negative) {
         h = &foundCorners[2];
         if (!h->negative) {
            h = &foundCorners[3];
```

```
                if (h->negative) {
                    vector<Point2> place;
                    for (int k = 0; k < foundCorners.size(); k++) {
                        place.push_back(foundCorners[k].corner);
                    }
                    corners.push_back(place);
                }
            }
        }
        else {
            h = &foundCorners[1];
            if (h->negative) {
                h = &foundCorners[2];
                if (h->negative) {
                    h = &foundCorners[3];
                    if (!h->negative) {
                        vector<Point2> place;
                        for (int k = 0; k < foundCorners.size(); k++) {
                            place.push_back(foundCorners[k].corner);
                        }
                        corners.push_back(place);
                    }
                }
            }
        }
        foundCorners.clear();
    }
}
```

Výpis 3: Část kódu hledající parkovací místo v kolekci úseček